

FILE HISTORY
US 6,691,139

PATENT: 6,691,139
INVENTORS: GANESH SHIVAJI
KAVANAPPILLIL RAMESH

TITLE: Recreation of archives at a disaster recovery site

APPLICATION NO: US2001774839A
FILED: 31 JAN 2001
ISSUED: 10 FEB 2004

COMPILED: 09 NOV 2023

U.S. UTILITY Patent Application

APPLICANTS

ISSUING CLASSIFICATION

☐ **TERMINAL
DISCLAIMER**

Form PTO-435A
(Rev. 8/88)

(Attached in pocket on right inside flap)

ISSUE FEE IN FILE

(FACE)

6,691,139

RECREATION OF ARCHIVES AT A DISASTER RECOVERY SITE

Transaction History

Date	Transaction Description
01-31-2001	Workflow - Drawings Finished
01-31-2001	Workflow - Drawings Matched with File at Contractor
01-31-2001	Information Disclosure Statement (IDS) Filed
01-31-2001	Information Disclosure Statement (IDS) Filed
01-31-2001	Initial Exam Team nn
03-13-2001	Correspondence Address Change
03-14-2001	Application Is Now Complete
03-14-2001	Notice Mailed--Application Incomplete--Filing Date Assigned
06-25-2001	Application Dispatched from OIPE
11-09-2001	Case Docketed to Examiner in GAU
12-13-2002	Case Docketed to Examiner in GAU
01-27-2003	Non-Final Rejection
01-28-2003	Mail Non-Final Rejection
04-28-2003	Response after Non-Final Action
05-29-2003	Date Forwarded to Examiner
08-01-2003	Final Rejection
08-05-2003	Mail Final Rejection (PTOL - 326)
10-09-2003	Response after Final Action
10-28-2003	Date Forwarded to Examiner
11-03-2003	Formal Drawings Required
11-03-2003	Notice of Allowance Data Verification Completed
11-04-2003	Mail Notice of Allowance
11-04-2003	Mail Formal Drawings Required
11-12-2003	Dispatch to Publications
12-02-2003	Dispatch to Publications
12-03-2003	Workflow - File Sent to Contractor
12-03-2003	Receipt into Pubs
12-04-2003	Issue Fee Payment Verified
12-04-2003	Issue Fee Payment Received
01-05-2004	Receipt into Pubs
01-07-2004	Application Is Considered Ready for Issue
01-09-2004	Receipt into Pubs
01-22-2004	Issue Notification Mailed
02-10-2004	Recordation of Patent Grant Mailed
02-10-2004	Patent Issue Date Used in PTA Calculation

CLASSIFICATION NOTES			
Examiner/ Classifier	Class	Date	Initials

FEE DETERMINATION			
O.I.P.E. CLASSIFIER		43	2/26/01
FORMALITY REVIEW	SL	1021	03/13/01
RESPONSE FORMALITY REVIEW	JK	035	06/25/01
1			

INDEX OF CLAIMS

✓		Rejected	N		Non-elected
=		Allowed	I		Interference
-	(Through numeral)...	Canceled	A		Appeal
÷		Restricted	O		Objected

Claim	Final	Original	Date
1	1	1	11/11/01
2	2	2	
3	3	3	
4	4	4	
5	5	5	
6	6	6	
7	7	7	
8	8	8	
9	9	9	
10	10	10	
11	11	11	
12	12	12	
13	13	13	
14	14	14	
15	15	15	
16	16	16	
17	17	17	
18	18	18	
19	19	19	
20	20	20	
21	21	21	
22	22	22	
23	23	23	
24	24	24	
25	25	25	
26	26	26	
27	27	27	
28	28	28	
29	29	29	
30	30	30	
31	31	31	
32	32	32	
33	33	33	
34	34	34	
35	35	35	
36	36	36	
37	37	37	
38	38	38	
39	39	39	
40	40	40	
41	41	41	
42	42	42	
43	43	43	
44	44	44	
45	45	45	
46	46	46	
47	47	47	
48	48	48	
49	49	49	
50	50	50	

Claim	Final	Original	Date
51	51	51	
52	52	52	
53	53	53	
54	54	54	
55	55	55	
56	56	56	
57	57	57	
58	58	58	
59	59	59	
60	60	60	
61	61	61	
62	62	62	
63	63	63	
64	64	64	
65	65	65	
66	66	66	
67	67	67	
68	68	68	
69	69	69	
70	70	70	
71	71	71	
72	72	72	
73	73	73	
74	74	74	
75	75	75	
76	76	76	
77	77	77	
78	78	78	
79	79	79	
80	80	80	
81	81	81	
82	82	82	
83	83	83	
84	84	84	
85	85	85	
86	86	86	
87	87	87	
88	88	88	
89	89	89	
90	90	90	
91	91	91	
92	92	92	
93	93	93	
94	94	94	
95	95	95	
96	96	96	
97	97	97	
98	98	98	
99	99	99	
100	100	100	

Claim	Final	Original	Date
101	101	101	
102	102	102	
103	103	103	
104	104	104	
105	105	105	
106	106	106	
107	107	107	
108	108	108	
109	109	109	
110	110	110	
111	111	111	
112	112	112	
113	113	113	
114	114	114	
115	115	115	
116	116	116	
117	117	117	
118	118	118	
119	119	119	
120	120	120	
121	121	121	
122	122	122	
123	123	123	
124	124	124	
125	125	125	
126	126	126	
127	127	127	
128	128	128	
129	129	129	
130	130	130	
131	131	131	
132	132	132	
133	133	133	
134	134	134	
135	135	135	
136	136	136	
137	137	137	
138	138	138	
139	139	139	
140	140	140	
141	141	141	
142	142	142	
143	143	143	
144	144	144	
145	145	145	
146	146	146	
147	147	147	
148	148	148	
149	149	149	
150	150	150	

If more than 150 claims or 10 actions
staple additional sheet here

(LEFT INSIDE)

ATTACH
DISK/FICHE/CD-ROM
ENVELOPE
HERE

(RIGHT INSIDE)

SEARCHED			
Class	Sub.	Date	Exmr.
707 ↓ 714	1-2 7,100- 101 200 204-205	1/23/03	Aene
711 ↓	3,100 143	1/23/03	Aue
update	Above	7/30/03	Aue
update	Above	11/1/03	Aene

[illegible]

INTERFERENCE SEARCHED			
Class	Sub.	Date	Exmr.
207	204	11.1.53	True
714	13 15	11.1.03	True

SAP - Exhibit 1002
SAP v. Valtrus et al. - IPR2025-00415
Page 7 of 201

WEST

Freeform Search

Database:

US Patents Full-Text Database

US Pre-Grant Publication Full-Text Database

JPO Abstracts Database

EPO Abstracts Database

Derwent World Patents Index

IBM Technical Disclosure Bulletins

Term:

ARCHIVE ADJ9 REDO ADJ9 LOG\$2

Display:

10

Documents in

Display Format:

-

Starting with Number

1

Generate:

☐ Hit List

☒ Hit Count

☐ Side by Side

☐ Image

Search

Clear

Help

Logout

Interrupt

Main Menu

Show 8 Numbers

Edit 8 Numbers

Preferences

Cases

Search History

DATE: Thursday, January 23, 2003 [Printable Copy](#) [Create Case](#)

<u>Set Name</u>	<u>Query</u>	<u>Hit Count</u>	<u>Set Name</u>
side by side			result set
	DB=USPT,PGPB,JPAB; PLUR=YES; OP=OR		
<u>L10</u>	ARCHIVE ADJ9 REDO ADJ9 LOG\$2	4	<u>L10</u>
<u>L9</u>	L7 AND ARCHIVE ADJ4 LOG\$1	2	<u>L9</u>
<u>L8</u>	L7 AND DATABASE AND ARCHIVE AND IMAGE NEAR9 DATABASE	2	<u>L8</u>
<u>L7</u>	MIRRORING NEAR9 DEVICE	543	<u>L7</u>
<u>L6</u>	L2 AND ARCHIV\$4 NEAR9 LOG\$1	1	<u>L6</u>
<u>L5</u>	L2 AND ARCHIVE ADJ9 LOG	1	<u>L5</u>
<u>L4</u>	L2 AND MIRRORING	1	<u>L4</u>
<u>L3</u>	L2 AND REDO NEAR9 LOG	1	<u>L3</u>
<u>L2</u>	ARCHIVE NEAR7 IMAG\$3 AND DATABASE AND LOG AND MIRRO\$5	25	<u>L2</u>
<u>L1</u>	ARCHIVE NEAR8 DELTA NEAR7 IMAG\$3	1	<u>L1</u>

END OF SEARCH HISTORY

Search Results -

Terms	Documents
delta adj5 imag\$2 and transaction adj4 redo adj4 log	4

Database:

- US Patents Full-Text Database

US Pre-Grant Publication Full-Text Database

JPO Abstracts Database

EPO Abstracts Database

Derwent World Patents Index

IBM Technical Disclosure Bulletins

Search:

0

Refine Search

Recall Text

Clear

Search History

DATE: Thursday, July 31, 2003 [Printable Copy](#) [Create Case](#)

<u>Set Name</u>	<u>Query</u>	<u>Hit Count</u>	<u>Set Name</u>
side by side			result set
DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=OR			
<u>L10</u>	delta adj5 imag\$2 and transaction adj4 redo adj4 log	4	<u>L10</u>
<u>L9</u>	L8 and site	6	<u>L9</u>
<u>L8</u>	active near5 transaction near6 redo and imag\$2 and separat\$3	11	<u>L8</u>
<u>L7</u>	active near5 transaction near6 redo and imag\$2 near4 separately	0	<u>L7</u>
<u>L6</u>	L5 and imag\$2 and primary	10	<u>L6</u>
<u>L5</u>	transaction adj4 redo adj4 log	34	<u>L5</u>
<u>L4</u>	transaction adj4 redo adj4 log and stanby	0	<u>L4</u>
<u>L3</u>	l2 and active adj4 transaction	4	<u>L3</u>
<u>L2</u>	transaction adj4 redo adj5 log and delta near7 image	4	<u>L2</u>
<u>L1</u>	transaction adj4 redo adj5 log and deltal near7 image	0	<u>L1</u>

END OF SEARCH HISTORY

<u>Set Name Query</u>		<u>Hit Count</u>	<u>Set Name</u>
side by side			result set
<i>DB=USPT,PGPB,JPAB; PLUR=YES; OP=OR</i>			
<u>L13</u>	l12 and imag\$4	2	<u>L13</u>
<u>L12</u>	l10 and mirror\$3	3	<u>L12</u>
<u>L11</u>	l10 and image	2	<u>L11</u>
<u>L10</u>	ARCHIVE ADJ9 REDO ADJ9 LOG\$2	4	<u>L10</u>
<u>L9</u>	L7 AND ARCHIVE ADJ4 LOG\$1	2	<u>L9</u>
<u>L8</u>	L7 AND DATABASE AND ARCHIVE AND IMAGE NEAR9 DATABASE	2	<u>L8</u>
<u>L7</u>	MIRRORING NEAR9 DEVICE	543	<u>L7</u>
<u>L6</u>	L2 AND ARCHIV\$4 NEAR9 LOG\$1	1	<u>L6</u>
<u>L5</u>	L2 AND ARCHIVE ADJ9 LOG	1	<u>L5</u>
<u>L4</u>	L2 AND MIRRORING	1	<u>L4</u>
<u>L3</u>	L2 AND REDO NEAR9 LOG	1	<u>L3</u>
<u>L2</u>	ARCHIVE NEAR7 IMAG\$3 AND DATABASE AND LOG AND MIRRO\$5	25	<u>L2</u>
<u>L1</u>	ARCHIVE NEAR8 DELTA NEAR7 IMAG\$3	1	<u>L1</u>

END OF SEARCH HISTORY



US006691139B2

(12) **United States Patent**
Ganesh et al.

(10) **Patent No.: US 6,691,139 B2**
(45) **Date of Patent: *Feb. 10, 2004**

(54) **RECREATION OF ARCHIVES AT A
DISASTER RECOVERY SITE**

6,035,412 A * 3/2000 Tamer et al. 714/6
2002/0091718 A1 * 7/2002 Bohannon et al. 707/202

(75) Inventors: **Shivaji Ganesh**, Milpitas, CA (US);
Ramesh Kavanappillil, Cupertino, CA
(US)

OTHER PUBLICATIONS

Roby Sherman, “Implementing an Automated Standby
Database”, Oracle Magazine, May 1999.

(73) Assignee: **Hewlett-Packard Development Co.,
Ltd.**, Houston, TX (US)

Biju Thomas, “Using Oracle8i’s New Log–Analysis Tool”,
Oracle Magazine, Sep. 1999.

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 303 days.

* cited by examiner

This patent is subject to a terminal dis-
claimer.

Primary Examiner—Alford Kindred

(21) Appl. No.: **09/774,839**

(57) **ABSTRACT**

(22) Filed: **Jan. 31, 2001**

(65) **Prior Publication Data**

US 2002/0103816 A1 Aug. 1, 2002

(51) **Int. Cl.**⁷ **G06F 11/00**; G06F 12/00

(52) **U.S. Cl.** **707/204**; 714/13; 714/15

(58) **Field of Search** 707/1–2, 7, 100–101,
707/200, 204–205; 714/13–15; 711/3, 100,
143

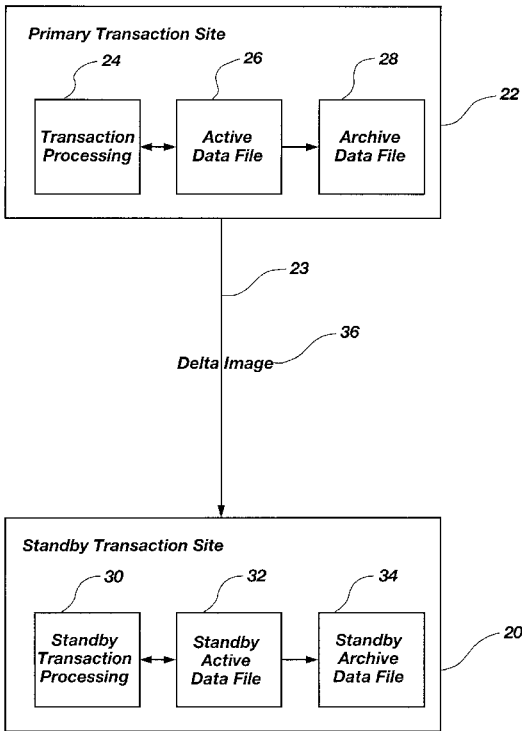
A method for creating a standby data processing system
located at a standby site, which is remotely located from a
primary site. The method comprises a first step of computing
a difference between an active data file and an archive data
file to produce a delta image at the primary site. Another step
is transmitting the active data file and the delta image,
separately in time sequence, from the primary site to a
standby site. A final step is combining the delta image with
the active data file at the standby site, to produce an archive
data file. An advantage of this method is the creation of a
standby data processing system, while using a lower band-
width to create this additional site.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,890,165 A * 3/1999 Boudrie et al. 707/202

20 Claims, 4 Drawing Sheets



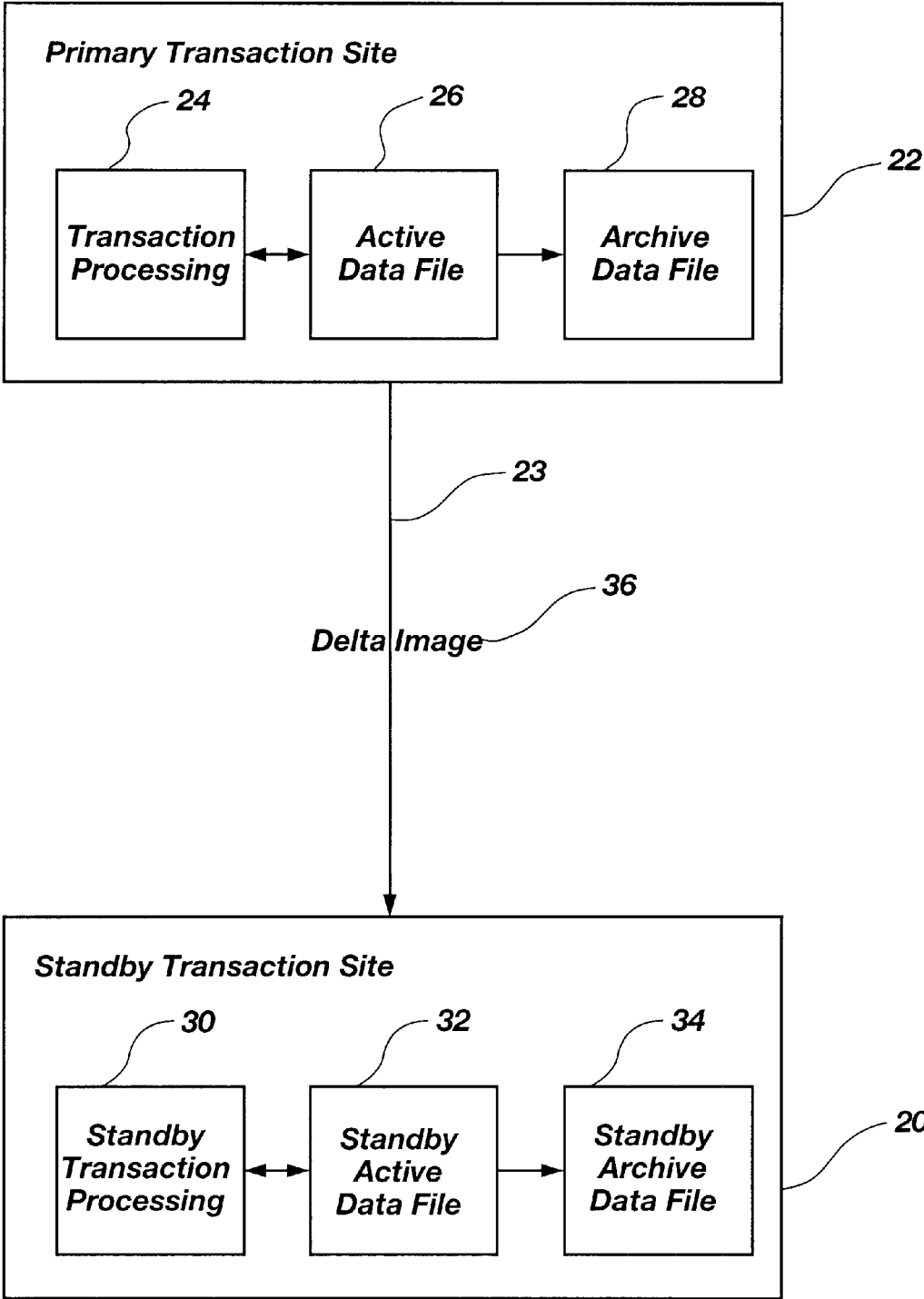


Fig. 1

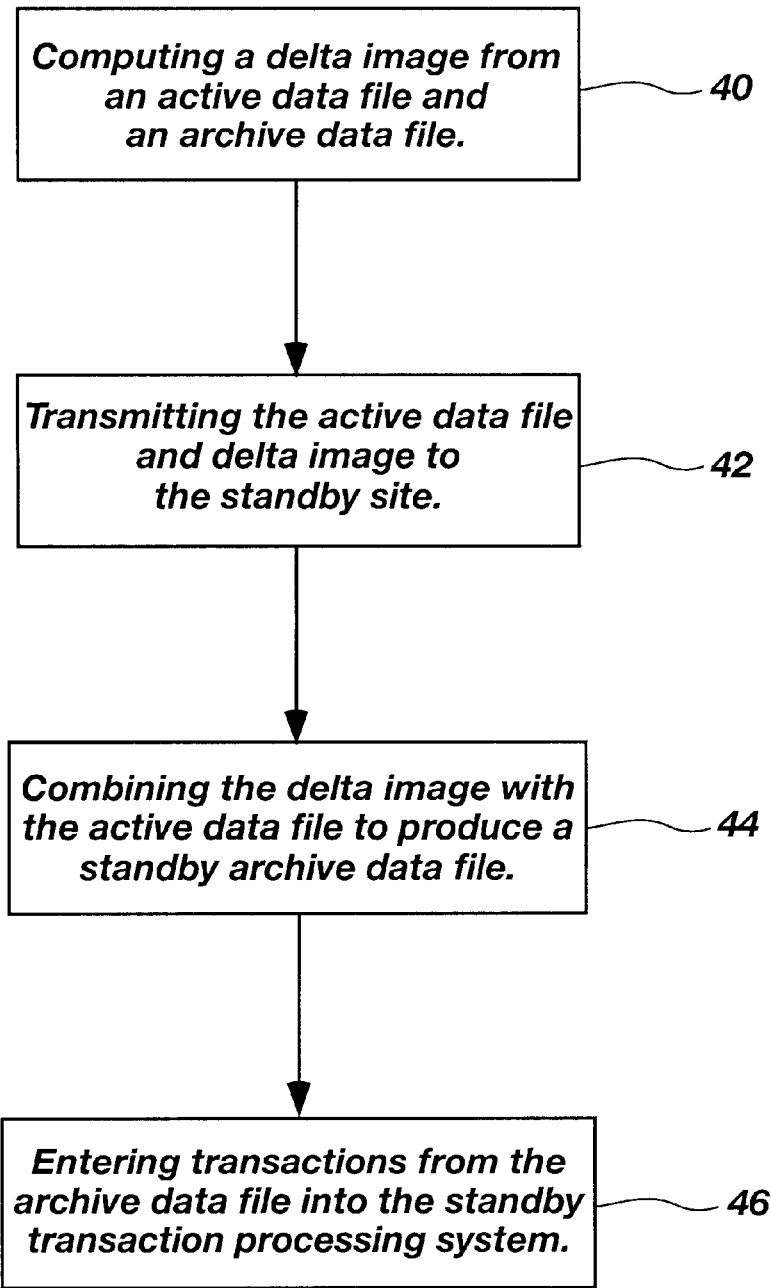


Fig. 2

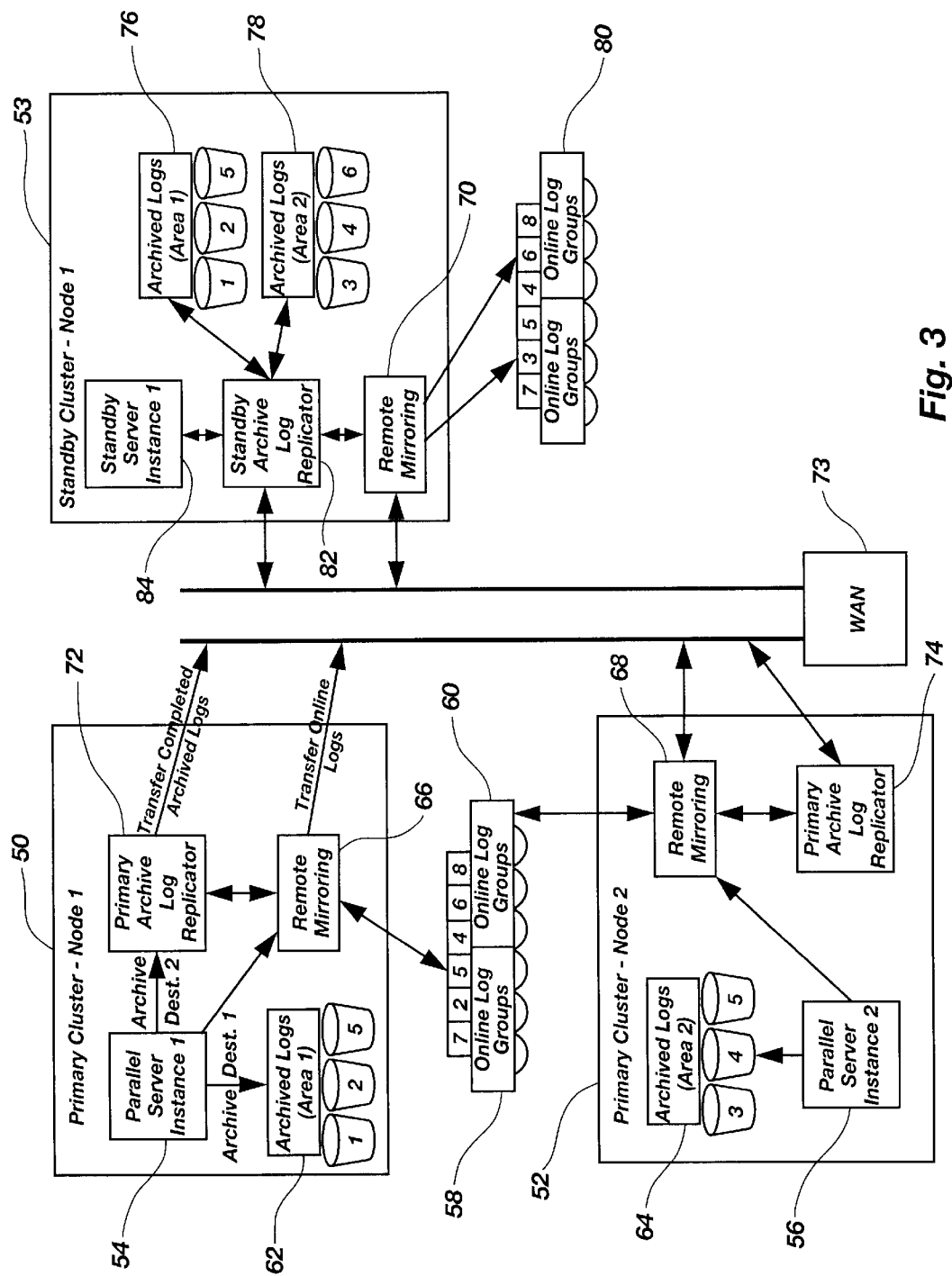


Fig. 3

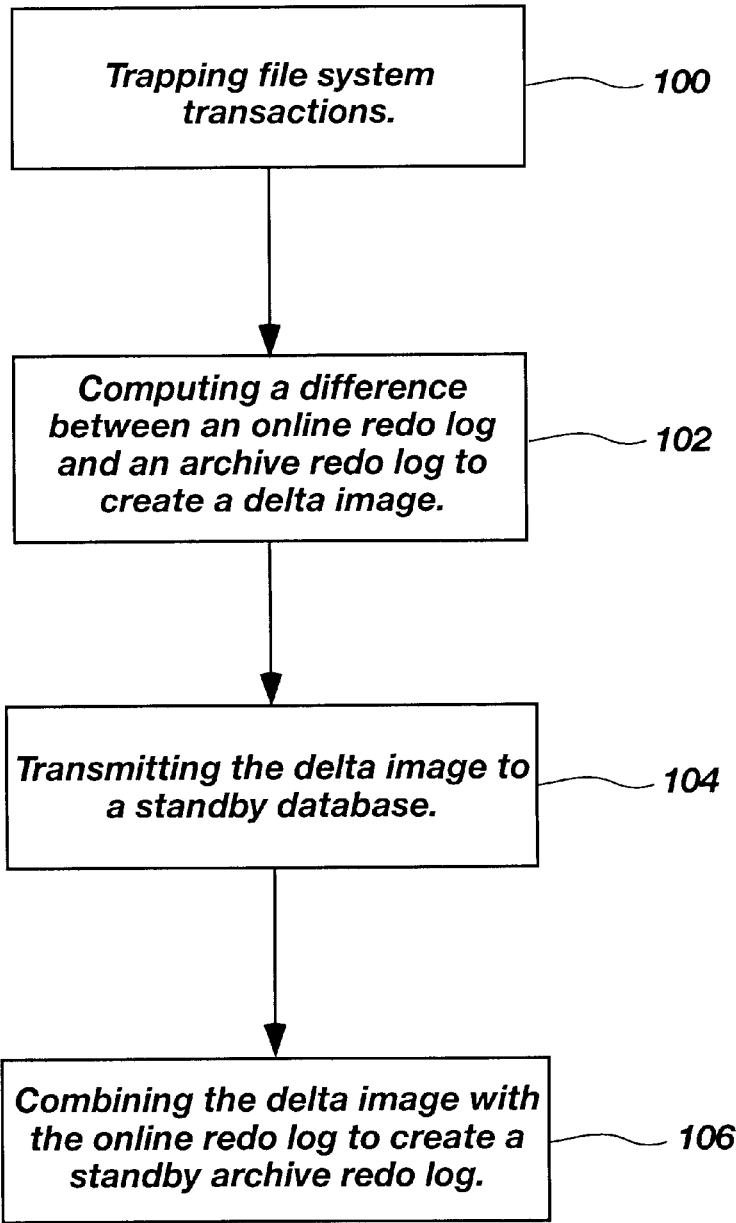


Fig. 4

1

RECREATION OF ARCHIVES AT A
DISASTER RECOVERY SITE

TECHNICAL FIELD

The present invention relates generally to disaster recovery for online computer systems. More particularly, the present invention relates to the creation of a standby data processing site at a remote location using reduced communication bandwidth mirroring.

BACKGROUND

In recent years, the use of networked computer software and the Internet has brought about a significant increase in the amount of network traffic and transactions performed by software applications residing on remote servers. More information is stored by these networked applications and in remote database applications than ever before. These applications process a large number of purchase transactions, credit card transactions, electronic mailing lists, email functions, data distribution, batch processing, etc. These systems contain very critical data and it must be backed up so that the information is not lost. Further, application end users also desire that networked applications and data should be available 24 hours a day and 7 days a week.

To provide robust services that are constantly available, computer systems must have redundant backup systems. It is inevitable that the primary system will fail on occasion. When the primary system fails, a backup system must be quickly available. A backup system can be located on-site with the primary system and then a secondary backup system can be located at a physically remote backup site. Having at least one backup system on-site is valuable because the networked application can immediately failover to that application, if the primary system becomes inoperative. This fast failover will be transparent to the user because it is local.

A second backup system at a remote site is desirable because it protects against catastrophic failure at the primary site. This is sometimes called the standby site. If the primary site is disabled by an extended power outage, fire, or flood then the remote standby system will be activated. A failover to an off-site standby system is slower, but it provides a maximum amount of protection against total system failure. Specifically, this type of fail safe system is valuable for applications that are connected to the Internet which need to be constantly available.

Many online, transaction based systems store more than one copy of their data. This is another safeguard against system failure or data corruption. The first copy is an original copy and the second is an archival copy. The archival copy is a backup but other times it is used for error checking purposes. Some systems can even store multiple backup copies.

For certain transactional systems, such as a database, an active transaction log is kept which tracks recent transactions. An archive log is then kept to store information from the active log after the active log has been filled or a certain time period has passed. To be able to mirror a transactional system between the primary site and the standby site both the active logs and then the archive logs must be transferred to the standby system. The active logs are then entered into or applied to the standby system which constantly keeps the standby system current.

In the event of a failure, the standby system is always ready to take over. Usually, the standby system is located in

2

another building or in a geographically remote area. This requires the active logs, archive logs, or backups to be transferred across a wide area network (WAN), which is relatively slow in comparison to a local area network (LAN) or a high speed cluster connection. As a result, the data sent from the primary site across the WAN to the standby site can cause a significant amount of network congestion. Especially where these files can contain hundreds of megabytes of data per file.

SUMMARY OF THE INVENTION

This invention provides a method for creating a standby data processing system located at a standby site, which is remotely located from a primary site. The method comprises a first step of computing a difference between an active data file and an archive data file to produce a delta image at the primary site. A second step is transmitting the active data file and the delta image, separately in time sequence, from the primary site to the standby site. A final step is combining the delta image with the active data file at the standby site, to produce a standby archive data file.

In accordance with one aspect of the present invention, the system includes a method for creating archive redo logs for a standby database at a remote location. The method includes the first step of trapping file system transactions using a primary archive log replicator to create an online redo log at the standby database. The next step is computing a difference between an online redo log and archive redo log in the primary database to create a delta image. Then the delta image is transmitted to a standby archive log replicator contained in the standby database.

Another step is combining the delta image with the online redo log in the standby database to create a standby archive redo log on the standby database. The final step is applying the updated archive redo log to the standby database.

Additional features and advantages of the invention will be set forth in the detailed description which follows, taken in conjunction with the accompanying drawings, which together illustrate by way of example, the features of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram illustrating the creation of an archive on a standby system;

FIG. 2 is a flow chart illustrating the method used to create archive files on a standby system;

FIG. 3 is a block diagram of a database system with a standby archive system that is created over a wide area network (WAN);

FIG. 4 is a flow chart illustrating the steps used to create archive files in a database system.

DETAILED DESCRIPTION

For the purposes of promoting an understanding of the invention, reference will now be made to the exemplary embodiments illustrated in the drawings, and specific language will be used to describe the same. It will nevertheless be understood that no limitation of the scope of the invention is thereby intended. Any alterations and further modifications of the inventive features illustrated herein, and any additional applications of the invention as illustrated herein, which would occur to one skilled in the relevant art and having possession of this disclosure, are to be considered within the scope of the invention.

FIG. 1 illustrates a device and method for creating archive data files from online data files at a standby site 20. An

3

advantage of this method is the creation of a standby data processing system located at a standby site, while using a lower network bandwidth to create archives at this additional site. This invention provides nearly a 50% savings in network bandwidth as compared to other remote mirroring solutions for transaction based systems. This is because it does not transfer the entire archive file system over the network.

The standby transaction site **20** (standby site) is remotely located from a primary transaction site **22** (primary site) to allow the system to recover from a catastrophic failure at the primary site. A communications link **23** connects the primary site to the standby site. The communications link is typically a wide area network (WAN), but a WAN is typically not as fast as a local area network (LAN) or a fast cluster connection. The communications link can also be a wireless link. The primary site includes a critical computer application which performs transaction processing **24**. The transaction processing contains a processing engine and a storage area to store the transactions processed. Transactional applications can include databases, web servers, e-commerce services, and similar transactional based services. Because the primary site contains a critical computer application, it is critical that the overall system does not completely fail. For example, the primary site might fail due to a fire or flood. Not only is it important to have a standby site, but both sites should be constantly synchronized. Then the standby site can take over if the primary site fails and this avoids the loss of data or transactions that were currently taking place. It also means that end users can still access the overall system.

As transactions take place at the primary site **22**, an active data file **26** or active log is kept of all the transactions that are performed. If the transaction processing fails at any point in time, then the active data file can be used to later re-apply those transactions. The active data file or active log has a limited size or it may be limited to a certain time period. When the active data file fills up, then it is archived to an archive data file **28**.

In a system that has a mirrored standby site **20**, the system periodically updates the standby active data file **32** or standby active log. As the standby active log is created, the transactions are rolled forward or applied to the standby transaction processing **30** so that the standby transaction processing contains a current image of what exists at the primary site. Conventional transaction based systems and databases include a managed standby solution that ships only the archive data file (after the active log is filled) over the WAN to the standby site. The problem with this approach is that the transactions that were not yet archived will be lost in a disaster and applications cannot implement a way for transactions to be 2-safe (i.e., reach both sites). In some cases, a third party mirroring system can be used to mirror the entire system or database using synchronous/asynchronous mirroring. The problem with this solution is that the mirroring of the entire database consumes a significant amount of bandwidth on a WAN which is already relatively slow.

The present invention has recognized that it is valuable to compute a difference between an active data file and an archive data file to produce a delta image **36** at the primary site. This delta image is useful because the archive data file is an archive of the active data file and there are large similarities between both files. As mentioned, it consumes a large amount of bandwidth to send the whole archive data file, so is more economical to send a smaller delta image. The delta image is computed by looking at differences in the

4

beginning or ending of the files. This is because large portions of the active data file will be the same as the archive data file. The difference in some situations may only be an archive header, but there can be other important differences. Alternatively, the system can also compare the files to see if there are blocks of additional information within the archive data file. These differences can also be incorporated with the delta image.

Once the delta image has been created, it is transmitted from the primary site to the standby site. The active data file **26** which was shipped to the standby site previously now becomes the standby active data file **32**. The delta image is then combined with the standby active data file at the standby site, to produce an archive data file **34**. The integration of the two files can be performed using overlay techniques, appending the delta image to the active data file, or applying a decompression algorithm to the delta image.

In an alternative embodiment of the device as shown in FIG. 1, the transaction processing may not contain any storage space and the active data file may be the primary storage system. The primary storage system can then be archived and the same delta image technique can be used to create the archive of the storage system on the standby system.

FIG. 2 is a flow chart illustrating the method used to create archive files on a standby system. The method comprises a first step of computing a difference between an active data file and an archive data file to produce a delta image at the primary site **40**. A second step is transmitting the active data file and the delta image, separately in time sequence, from the primary site to the standby site **42**. The next step is combining the delta image with the active data file at the standby site, to produce a standby archive data file **44**. A follow-up step is entering transactions from the archive data file into the standby transaction processing system **46**.

FIG. 3 is a block diagram illustrating one embodiment of this invention. This embodiment describes the use of the present invention to create archives for a database system. The figure also illustrates the interactions involved in replicating archive files across a WAN to a remote standby site or cluster.

The system has a first primary cluster node **50** and a second primary cluster node **52** and these two nodes form a database cluster. A disk subsystem can be shared by these nodes and the nodes can be connected by a high speed local link. Each node has a separate instance of a parallel database server **54**, **56** and each node will have its own copy of the online redo logs **58**, **60** (i.e., an active data file). As is well known by those skilled in the art, redo log files store all the information needed to perform database recovery and they record every change made to the database structure. When the online redo logs are filled they are then archived and transferred into the archive log areas **62**, **64** (i.e., archive data file).

The steps involved in archived redo log replication will now be discussed in further detail. First, the parallel database servers **54**, **56** fill up the online redo logs **58**, **60** when the transactions commit. Remote mirroring devices **66**, **68** trap the disk subsystem writes to the online logs. This trapped information is then sent to the standby remote mirroring device **70** through the remote mirroring or replication transport. A WAN **73** is used to transfer the information between the primary site and the standby site.

The archive systems of the parallel database server **54**, **56** archive the filled up online redo logs to two destinations. One archive destination is a local high performance, quick

5

recovery, file system **62** (e.g., VXFS). The other destination is a special file system generally called the archive replicator file system. All opens/writes and closes to this file system are trapped by the primary archive log replicator **72, 74**. The archive log replicator ships newly generated online redo log

files across to the standby site and asks the standby database to apply it in such a fashion that 2-safeness and data integrity are preserved at all times.

In this invention, the archive files generated at the primary site will be sent to the standby site as a delta image. This is a technique wherein only the difference between the archive log file and the corresponding online redo file is sent across to the standby site. The standby site can recreate the standby archive file using this difference and the mirrored online redo log.

In order to reliably build the archive file at the standby site, and to prevent over-shoots and under-shoots of archive logs versus the online logs (across all types of failures and race condition scenarios), an effective remote mirroring technology must be used. Over-shoot refers to the situation where there can be a gap between what exists in the active logs and the archive logs on the standby site. In contrast, an under-shoot refers to the situation where there are transactions in the archive logs that are ahead in time and sequence to what is available in the active logs. Either of these situations require a system administrator to perform a reset logs when the standby site needs to take over, which causes more data loss (and operator intervention) as the transactions in the active logs are lost on the standby site.

In this mirroring approach, the online redo log is mirrored using asynchronous remote mirroring technology **66, 70**. The remote mirroring component can be a UNIX device driver that works by trapping all operating system writes for a storage device and mirroring these writes across disks on remote systems. The most effective type of remote mirroring component preserves write ordering across all writes from the primary cluster to the standby cluster. It should be pointed out that the delta image may use remote procedure calls (RPC) to ship the delta image to the standby site.

The remote mirroring technology is used to remotely mirror online redo logs from the primary cluster running the parallel database server. The primary archive log replicator **72, 74** traps all the archive writes on all the nodes of the primary site. It then computes the difference between the archive redo log and the corresponding online redo log (which can differ only by a few bytes) and sends only this difference over the network to the standby site archive file replicator **82**. The delta image difference in some situations may only be an archive header but there can be other important differences.

The standby archive log replicator **82** checks with the remote mirroring component **70** to confirm whether the corresponding online redo log has been completely mirrored. The communication between the standby archive log replicator and the remote mirroring device is completed using replication handles. Replication handles help track the progress of the mirror writes on standby site as compared to the writes on the primary site. The replication handles are sequence numbers which the remote mirroring component uses to sequence every write on the primary site and to track progress on the standby site.

The archive log replicator **82** at the standby site coordinates with the standby site's remote mirroring component **70** and re-creates the archive redo log **76, 78** from the corresponding online redo log(s) **80** which are mirrored on the standby site.

6

One method of handling the delta image and creation of the archive log files **76, 78** on the standby site will now be described. After the standby remote mirroring device **70** indicates the completion of mirroring (through the replication handles), the standby archive log replicator copies the standby online redo log corresponding to the delta archive image and overlays this delta image to create the archive. This new archive is stored in a temporary place until the primary archive log replicator **72** sends a message indicating archive completion at the primary. The primary archive log replicator signals completion of the new standby archive redo log by checking with the database system using database queries for archive completion.

After the creation of the standby archive redo log **76**, the standby archive log replicator **82** checks with the corresponding database (through queries) to decide whether the archiving is complete. If the archive log is complete, the standby archive file replicator proceeds with applying the completed archive to the standby database sever **84**. The database does this by creating a link to the file in the archive destination where the standby database is looking for files. Another advantage of this invention is that it prevents over-shoots, under-shoots, or out-of-order writes. This ensures that the standby site can be used as the primary site when necessary without the need for resetting the standby logs. As mentioned, resetting the active logs causes data loss because the data in the active logs is ignored by the database system.

If for some reason (say due to WAN outages or high congestion) the archive log delta images cannot be shipped and the number of unarchived but the completed online redo logs reach a threshold, the primary archive replicator enters catch-up mode. The threshold can be defined as when the a spool buffer in the remote mirroring device is filled or when the archiving becomes slow enough (due to WAN delay or outages) that there is a risk of the primary database system blocking because it cannot archive the active log files. In catch-up mode, the delta images are discarded and the whole completed archive redo log images are shipped asynchronously. Of course, the primary parallel database server continues to generate archives during the network outage. However, shipping delta images to the second destination or the standby site is deferred, until the primary archive file replicator is able to ship all the pending archive redo logs to the standby site and re-enter normal mode.

On entering catch-up mode, the primary archive replicator automatically asks the remote mirroring device to pause the remote mirroring. This ensures that the online never over-shoots the archives. Including a catch-up mode allows this system to withstand WAN outages of long duration without sacrificing the operation performance of the primary site.

FIG. 4 is a flow chart illustrating a summary of the steps used to create archive files in a database system. The method for creating archive redo logs for a standby database at a remote location includes several steps. The first step is trapping file system transactions or active log writes using a remote mirroring device capable of mirroring the log writes to a standby site **100**. This creates an online redo log at the standby site. The next step is computing a difference between an online redo log and archive redo log to create a delta image at a primary database **102**. Another step is transmitting the delta image to a standby archive log replicator contained in the standby database **104**. The last step is combining the delta image with the online redo log to create a standby archive redo log on the standby database **106**.

It is to be understood that the above-described arrangements are only illustrative of the application of the principles

7

of the present invention. Numerous modifications and alternative arrangements may be devised by those skilled in the art without departing from the spirit and scope of the present invention and the appended claims are intended to cover such modifications and arrangements. Thus, while the present invention has been shown in the drawings and fully described above with particularity and detail in connection with what is presently deemed to be the most practical and preferred embodiment(s) of the invention, it will be apparent to those of ordinary skill in the art that numerous modifications, including, but not limited to, variations in function, manner of operation, use, and specific implementation may be made, without departing from the concepts of the invention as set forth in the claims.

What is claimed is:

1. A method for creating a standby data processing system located at a standby site, which is remotely located from a primary site, comprising the steps of:

- (a) computing a difference between an active transaction redo log and an archive transaction redo log to produce a delta image at the primary site;
- (b) transmitting the active transaction redo log and the delta image, separately in time sequence, from the primary site to the standby site; and
- (c) combining the delta image with the active transaction redo log at the standby site, to produce a standby archive transaction redo log.

2. A method as in claim 1, further comprising the step of storing the standby archive transaction redo log on the standby site.

3. A method as in claim 1, further comprising the step of storing transactions in the standby active transaction redo log as they are trapped at the primary site.

4. A method as in claim 1, further comprising the step of entering transactions from the archive transaction redo log into the standby transaction processing system.

5. A method for creating archive redo logs for a standby database at a remote location, comprising the steps of:

- (a) trapping active log writes using a remote mirroring device that is capable of mirroring the log writes to a standby site, to create an online redo log at the standby site;
- (b) computing a difference between an online redo log and archive redo log to create a delta image at a primary database, wherein the delta image includes an archive header;
- (c) transmitting the delta image to a standby archive log replicator contained in the standby database; and
- (d) combining the delta image and archive header with the online redo log to create a standby archive redo log on the standby database.

6. A method as in claim 5, further comprising the step of applying the standby archive redo log by creating links to the standby archive redo log.

7. A method as in claim 5, further comprising the step of transmitting the delta image to a standby archive log replicator using a first remote mirroring device in the primary database and a second remote mirroring device in the standby database.

8. A method as in claim 7, further comprising the step of using replication handles to track the progress of mirror writes on the standby database as compared to the primary database.

9. A method as in claim 8, further comprising the step of checking the primary database queries using the archive log replicator to determine whether archiving is complete.

8

10. A method as in claim 5, further comprising the step of shipping an entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously after archiving becomes slow enough that there is a risk of primary database blocking.

11. A method as in claim 10, further comprising the step of shipping an entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously when a spool buffer in a remote mirroring system has been filled.

12. A method for creating archive redo logs from a primary database, for a standby database at a remote location, comprising the steps of:

- (a) trapping file system transactions to create an online redo log at the standby database;
- (b) computing a difference between an online redo log and archive redo log in the primary database to find the delta image of the archive redo log;
- (c) transmitting the delta image to a standby database; and
- (d) combining the delta image with the online redo log in the standby database to create standby archive redo log on the standby database.

13. A method as in claim 12, further comprising the step of applying the standby archive redo log by creating a link between the standby database and the standby archive redo log.

14. A method as in claim 12, further comprising the step of transmitting the delta image to a standby archive log replicator using a first remote mirroring device in the primary database and a second remote mirroring device in the standby database.

15. A method as in claim 14, further comprising the step of using replication handles to track the progress of mirror writes on the standby database as compared to the primary database.

16. A method as in claim 12, further comprising the step of checking the primary database with database queries using the archive log replicator to determine whether archiving is complete.

17. A method as in claim 12, further comprising the step of shipping an entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously after archiving becomes slow enough that there is a risk of the primary database blocking.

18. A method as in claim 12, further comprising the step of shipping the entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously when a spool buffer in a remote mirroring system has been filled.

19. A system for creating a standby data processing system located at a standby site, which is remotely located from a primary site, comprising the steps of:

- means for computing a difference between an active data file and an archive data file to produce a delta image at the primary site;
- means for transmitting the active data file and the delta image, separately in time sequence, from the primary site to the standby site; and
- means for combining the delta image with the active data file at the standby site, to produce a standby archive data file.

20. An article of manufacture comprising:

- a computer usage medium having computer readable program code means embodied therein for creating archive redo logs for a standby database at a remote location, the computer readable program code means in said article of manufacture comprising:

9

computer readable program code means for trapping active log writes using a remote mirroring device that is capable of mirroring the log writes to a standby site, to create an online redo log at the standby site;
computer readable program code means for computing a difference between an online redo log and archive redo log to create a delta image at a primary data-base;

5

10

computer readable program code means for transmitting the delta image to a standby archive log replicator contained in the standby database; and
computer readable program code means for combining the delta image with the online redo log to create a standby archive redo log on the standby database.

* * * * *



UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
UNITED STATES PATENT AND TRADEMARK OFFICE
WASHINGTON, D.C. 20231
www.uspto.gov



Bib Data Sheet

CONFIRMATION NO. 6266

SERIAL NUMBER 09/774,839	FILING DATE 01/31/2001 RULE	CLASS 711 207	GROUP ART UNIT 2185 2172	ATTORNEY DOCKET NO. 10006924-1
APPLICANTS Shivaji Ganesh, Milpitas, CA; Ramesh Kavanappillil, Cupertino, CA;				
** CONTINUING DATA *****				
** FOREIGN APPLICATIONS *****				
IF REQUIRED, FOREIGN FILING LICENSE GRANTED ** 03/13/2001				
Foreign Priority claimed ☐ yes ☒ no 35 USC 119 (a-d) conditions met ☐ yes ☒ no ☐ Met after Allowance		STATE OR COUNTRY CA	SHEETS DRAWING 4	TOTAL CLAIMS 22 20
Verified and Acknowledged Examiner's Signature Initials		INDEPENDENT CLAIMS 5		
ADDRESS HEWLETT-PACKARD COMPANY Intellectual Property Administration P.O. Box 272400 Fort Collins ,CO 80527-2400				
TITLE Recreation of archives at a disaster recovery site				
FILING FEE RECEIVED 1036	FEES: Authority has been given in Paper No. _____ to charge/credit DEPOSIT ACCOUNT No. _____ for following:		☐ All Fees ☐ 1.16 Fees (Filing) ☐ 1.17 Fees (Processing Ext. of time) ☐ 1.18 Fees (Issue) ☐ Other _____ ☐ Credit	

PATENT APPLICATION SERIAL NO. _____

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

05/05/1991	MAR911	00000024	082025	09774839
01	00.00	710.00	CH	
02	00.00	160.00	CH	
03	00.00	36.00	CH	

PTO-1556
(5/87)

*U.S. GPO: 1999-459-082/19144

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P. O. Box 272400
Fort Collins, Colorado 80527-2400

11003 U.S. PTO



01/31/01

02-01-01

PATENT APPLICATION

ATTOR

DOCKET NO. 10006924-1

IN THE U.S. PATENT AND TRADEMARK OFFICE
Patent Application Transmittal Letter

COMMISSIONER FOR PATENTS
Washington, D.C. 20231

Sir:

Transmitted herewith for filing under 37 CFR 1.53(b) is a(n): ☒ Utility ☐ Design

☒ original patent application,

☐ continuation-in-part application

INVENTOR(S): S. Ganesh and R. Kavanappillil

TITLE: RECREATION OF ARCHIVES AT A DISASTER RECOVERY SITE

Enclosed are:

- ☐ The Declaration and Power of Attorney. ☐ signed ☐ unsigned or partially signed
☒ 4 sheets of drawings (one set) ☐ Associate Power of Attorney
☐ Form PTO-1449 ☒ Information Disclosure Statement and Form PTO-1449
☐ Priority document(s) ☐ (Other) (fee \$)

CLAIMS AS FILED BY OTHER THAN A SMALL ENTITY				
(1) FOR	(2) NUMBER FILED	(3) NUMBER EXTRA	(4) RATE	(5) TOTALS
TOTAL CLAIMS	22 — 20	2	X \$18	\$ 36
INDEPENDENT CLAIMS	5 — 3	2	X \$80	\$ 160
ANY MULTIPLE DEPENDENT CLAIMS	0		\$270	\$ 0
BASIC FEE: Design (\$320.00); Utility (\$710.00)				\$ 710
TOTAL FILING FEE				\$ 906
OTHER FEES				\$
TOTAL CHARGES TO DEPOSIT ACCOUNT				\$ 906

Charge \$ 906 to Deposit Account 08-2025. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16, 1.17, 1.19, 1.20 and 1.21. A duplicate copy of this sheet is enclosed.

"Express Mail" label no. EL163092522US

Date of Deposit 1/31/01

I hereby certify that this is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to: Commissioner for Patents, Washington, D.C. 20231.

By Steve M. Perry

Typed Name: Steve M. Perry

Respectfully submitted,

S. Ganesh and R. Kavanappillil

By Steve M. Perry

Steve M. Perry

Attorney/Agent for Applicant(s)

Reg. No. 45,357

Date: 1/31/01

Telephone No.: (801) 566-6633

Express Mail Label No. EL 163 092 522 US

Patent Application
HP Docket No. 10006924

United States Patent Application

of

Shivaji Ganesh and

Ramesh Kavanappillil

for

Recreation of Archives

at a Disaster Recovery Site

0974839.013101
PCT/IN/0684260

TO THE COMMISSIONER OF PATENTS AND TRADEMARKS:

Your petitioners, **Shivaji Ganesh**, citizen of the United States of America, whose residence and postal mailing address is 1848 Wellington Drive, Milpitas, California, 95035; and **Ramesh Kavanappillil**, citizen of India, whose residence and postal mailing address is 19608 Pruneridge Avenue, #9204, Cupertino, California 95014, prays that letters patent may be granted to them as the inventors of **Recreation of Archives at a Disaster Recovery Site** as set forth in the following specification.

TECHNICAL FIELD

The present invention relates generally to disaster recovery for online computer systems. More particularly, the present invention relates to the creation of a standby data processing site at a remote location using reduced communication bandwidth mirroring.

BACKGROUND

In recent years, the use of networked computer software and the Internet has brought about a significant increase in the amount of network traffic and transactions performed by software applications residing on remote servers. More information is stored by these networked applications and in remote database applications than ever before. These applications process a large number of purchase transactions, credit card transactions, electronic mailing lists, email functions, data distribution, batch processing, etc. These systems contain very critical data and it must be backed up so that the information is not lost. Further, application end users also desire that networked applications and data should be available 24 hours a day and 7 days a week.

To provide robust services that are constantly available, computer systems must have redundant backup systems. It is inevitable that the primary system will fail on occasion. When the primary system fails, a backup system must be quickly available. A backup system can be located on-site with the primary system and then a secondary backup system can be located at a physically remote backup site. Having at least one backup system on-site is valuable because the networked application can immediately failover to that application, if the primary system becomes inoperative. This fast failover will be transparent to the user because it is local.

0074829.04104
FOI b6 b7C b7D

A second backup system at a remote site is desirable because it protects against catastrophic failure at the primary site. This is sometimes called the standby site. If the primary site is disabled by an extended power outage, fire, or flood then the remote standby system will be activated. A failover to an off-site standby system is slower, but it provides a maximum amount of protection against total system failure. Specifically, this type of fail safe system is valuable for applications that are connected to the Internet which need to be constantly available.

Many online, transaction based systems store more than one copy of their data. This is another safeguard against system failure or data corruption. The first copy is an original copy and the second is an archival copy. The archival copy is a backup but other times it is used for error checking purposes. Some systems can even store multiple backup copies.

For certain transactional systems, such as a database, an active transaction log is kept which tracks recent transactions. An archive log is then kept to store information from the active log after the active log has been filled or a certain time period has passed. To be able to mirror a transactional system between the primary site and the standby site both the active logs and then the archive logs must be transferred to the standby system. The active logs are then entered into or applied to the standby system which constantly keeps the standby system current.

In the event of a failure, the standby system is always ready to take over. Usually, the standby system is located in another building or in a geographically remote area. This requires the active logs, archive logs, or backups to be transferred across a wide area network (WAN), which is relatively slow in comparison to a local area network (LAN) or a high speed cluster connection. As a result, the data sent from the primary site across the WAN to the

standby site can cause a significant amount of network congestion. Especially where these files can contain hundreds of megabytes of data per file.

SUMMARY OF THE INVENTION

This invention provides a method for creating a standby data processing system located at a standby site, which is remotely located from a primary site. The method comprises a first step of computing a difference between an active data file and an archive data file to produce a delta image at the primary site. A second step is transmitting the active data file and the delta image, separately in time sequence, from the primary site to the standby site. A final step is combining the delta image with the active data file at the standby site, to produce a standby archive data file.

In accordance with one aspect of the present invention, the system includes a method for creating archive redo logs for a standby database at a remote location. The method includes the first step of trapping file system transactions using a primary archive log replicator to create an online redo log at the standby database. The next step is computing a difference between an online redo log and archive redo log in the primary database to create a delta image. Then the delta image is transmitted to a standby archive log replicator contained in the standby database.

Another step is combining the delta image with the online redo log in the standby database to create a standby archive redo log on the standby database. The final step is applying the updated archive redo log to the standby database.

Additional features and advantages of the invention will be set forth in the detailed description which follows, taken in conjunction with the accompanying drawings, which together illustrate by way of example, the features of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram illustrating the creation of an archive on a standby system;

FIG. 2 is a flow chart illustrating the method used to create archive files on a standby system;

FIG. 3 is a block diagram of a database system with a standby archive system that is created over a wide area network (WAN);

FIG. 4 is a flow chart illustrating the steps used to create archive files in a database system.

DETAILED DESCRIPTION

For the purposes of promoting an understanding the invention, reference will now be made to the exemplary embodiments illustrated in the drawings, and specific language will be used to describe the same. It will nevertheless be understood that no limitation of the scope of the invention is thereby intended. Any alterations and further modifications of the inventive features illustrated herein, and any additional applications of the invention as illustrated herein, which would occur to one skilled in the relevant art and having possession of this disclosure, are to be considered within the scope of the invention.

FIG. 1 illustrates a device and method for creating archive data files from online data files at a standby site 20. An advantage of this method is the creation of a standby data processing system located at a standby site, while using a lower network bandwidth to create archives at this additional site. This invention provides nearly a 50% savings in network bandwidth as compared to other remote mirroring solutions for transaction based systems. This is because it does not transfer the entire archive file system over the network.

The standby transaction site 20 (standby site) is remotely located from a primary transaction site 22 (primary site) to allow the system to recover from a catastrophic failure at the primary site. A communications link 23 connects the primary site to the standby site. The communications link is typically a wide area network (WAN), but a WAN is typically not as fast as a local area network (LAN) or a fast cluster connection. The communications link can also be a wireless link. The primary site includes a critical computer application which performs transaction processing 24. The transaction processing contains a processing engine and a storage area to store the transactions processed. Transactional applications can include databases, web servers, e-commerce services, and similar transactional based services. Because the primary site contains a critical computer application, it is critical that the overall system does not completely fail. For example, the primary site might fail due to a fire or flood. Not only is it important to have a standby site, but both sites should be constantly synchronized. Then the standby site can take over if the primary site fails and this avoids the loss of data or transactions that were currently taking place. It also means that end users can still access the overall system.

As transactions take place at the primary site 22, an active data file 26 or active log is kept of all the transactions that are performed. If the transaction processing fails at any point in time, then the active data file can be used to later re-apply those transactions. The active

data file or active log has a limited size or it may be limited to a certain time period. When the active data file fills up, then it is archived to an archive data file 28.

In a system that has a mirrored standby site 20, the system periodically updates the standby active data file 32 or standby active log. As the standby active log is created, the transactions are rolled forward or applied to the standby transaction processing 30 so that the standby transaction processing contains a current image of what exists at the primary site. Conventional transaction based systems and databases include a managed standby solution that ships only the archive data file (after the active log is filled) over the WAN to the standby site. The problem with this approach is that the transactions that were not yet archived will be lost in a disaster and applications cannot implement a way for transactions to be 2-safe (i.e., reach both sites). In some cases, a third party mirroring system can be used to mirror the entire system or database using synchronous/asynchronous mirroring. The problem with this solution is that the mirroring of the entire database consumes a significant amount of bandwidth on a WAN which is already relatively slow.

The present invention has recognized that it is valuable to compute a difference between an active data file and an archive data file to produce a delta image 36 at the primary site. This delta image is useful because the archive data file is an archive of the active data file and there are large similarities between both files. As mentioned, it consumes a large amount of bandwidth to send the whole archive data file, so is more economical to send a smaller delta image. The delta image is computed by looking at differences in the beginning or ending of the files. This is because large portions of the active data file will be the same as the archive data file. The difference in some situations may only be an archive header, but there can be other important differences. Alternatively, the system can also compare the files

to see if there are blocks of additional information within the archive data file. These differences can also be incorporated with the delta image.

Once the delta image has been created, it is transmitted from the primary site to the standby site. The active data file 26 which was shipped to the standby site previously now becomes the standby active data file 32. The delta image is then combined with the standby active data file at the standby site, to produce an archive data file 34. The integration of the two files can be performed using overlay techniques, appending the delta image to the active data file, or applying a decompression algorithm to the delta image.

In an alternative embodiment of the device as shown in FIG. 1, the transaction processing may not contain any storage space and the active data file may be the primary storage system. The primary storage system can then be archived and the same delta image technique can be used to create the archive of the storage system on the standby system.

FIG. 2 is a flow chart illustrating the method used to create archive files on a standby system. The method comprises a first step of computing a difference between an active data file and an archive data file to produce a delta image at the primary site 40. A second step is transmitting the active data file and the delta image, separately in time sequence, from the primary site to the standby site 42. The next step is combining the delta image with the active data file at the standby site, to produce a standby archive data file 44. A follow-up step is entering transactions from the archive data file into the standby transaction processing system 46.

FIG. 3 is a block diagram illustrating one embodiment of this invention. This embodiment describes the use of the present invention to create archives for a database system. The figure also illustrates the interactions involved in replicating archive files across a WAN to a remote standby site or cluster.

The system has a first primary cluster node 50 and a second primary cluster node 52 and these two nodes form a database cluster. A disk subsystem can be shared by these nodes and the nodes can be connected by a high speed local link. Each node has a separate instance of a parallel database server 54, 56 and each node will have its own copy of the online redo logs 58, 60 (i.e., an active data file). As is well known by those skilled in the art, redo log files store all the information needed to perform database recovery and they record every change made to the database structure. When the online redo logs are filled they are then archived and transferred into the archive log areas 62, 64 (i.e., archive data file).

The steps involved in archived redo log replication will now be discussed in further detail. First, the parallel database servers 54, 56 fill up the online redo logs 58, 60 when the transactions commit. Remote mirroring devices 66, 68 trap the disk subsystem writes to the online logs. This trapped information is then sent to the standby remote mirroring device 70 through the remote mirroring or replication transport. A WAN 73 is used to transfer the information between the primary site and the standby site.

The archive systems of the parallel database server 54, 56 archive the filled up online redo logs to two destinations. One archive destination is a local high performance, quick recovery, file system 62 (e.g., VXFS). The other destination is a special file system generally called the archive replicator file system. All opens/writes and closes to this file system are trapped by the primary archive log replicator 72, 74. The archive log replicator ships newly generated online redo log files across to the standby site and asks the standby database to apply it in such a fashion that 2-safeness and data integrity are preserved at all times.

In this invention, the archive files generated at the primary site will be sent to the standby site as a delta image. This is a technique wherein only the difference between the archive log file and the corresponding online redo file is sent across to the standby site. The

standby site can recreate the standby archive file using this difference and the mirrored online redo log.

In order to reliably build the archive file at the standby site, and to prevent over-shoots and under-shoots of archive logs versus the online logs (across all types of failures and race condition scenarios), an effective remote mirroring technology must be used. Over-shoot refers to the situation where there can be a gap between what exists in the active logs and the archive logs on the standby site. In contrast, an under-shoot refers to the situation where there are transactions in the archive logs that are ahead in time and sequence to what is available in the active logs. Either of these situations require a system administrator to perform a reset logs when the standby site needs to take over, which causes more data loss (and operator intervention) as the transactions in the active logs are lost on the standby site.

In this mirroring approach, the online redo log is mirrored using asynchronous remote mirroring technology 66, 70. The remote mirroring component can be a UNIX device driver that works by trapping all operating system writes for a storage device and mirroring these writes across disks on remote systems. The most effective type of remote mirroring component preserves write ordering across all writes from the primary cluster to the standby cluster. It should be pointed out that the delta image may use remote procedure calls (RPC) to ship the delta image to the standby site.

The remote mirroring technology is used to remotely mirror online redo logs from the primary cluster running the parallel database server. The primary archive log replicator 72, 74 traps all the archive writes on all the nodes of the primary site. It then computes the difference between the archive redo log and the corresponding online redo log (which can differ only by a few bytes) and sends only this difference over the network to the standby site

archive file replicator 82. The delta image difference in some situations may only be an archive header but there can be other important differences.

The standby archive log replicator 82 checks with the remote mirroring component 70 to confirm whether the corresponding online redo log has been completely mirrored. The communication between the standby archive log replicator and the remote mirroring device is completed using replication handles. Replication handles help track the progress of the mirror writes on standby site as compared to the writes on the primary site. The replication handles are sequence numbers which the remote mirroring component uses to sequence every write on the primary site and to track progress on the standby site.

The archive log replicator 82 at the standby site coordinates with the standby site's remote mirroring component 70 and re-creates the archive redo log 76, 78 from the corresponding online redo log(s) 80 which are mirrored on the standby site.

One method of handling the delta image and creation of the archive log files 76, 78 on the standby site will now be described. After the standby remote mirroring device 70 indicates the completion of mirroring (through the replication handles), the standby archive log replicator copies the standby online redo log corresponding to the delta archive image and overlays this delta image to create the archive. This new archive is stored in a temporary place until the primary archive log replicator 72 sends a message indicating archive completion at the primary. The primary archive log replicator signals completion of the new standby archive redo log by checking with the database system using database queries for archive completion.

After the creation of the standby archive redo log 76, the standby archive log replicator 82 checks with the corresponding database (through queries) to decide whether the archiving is complete. If the archive log is complete, the standby archive file replicator

0974839.030
T07E70"6E8h/260

proceeds with applying the completed archive to the standby database sever 84. The database does this by creating a link to the file in the archive destination where the standby database is looking for files. Another advantage of this invention is that it prevents over-shoots, under-shoots, or out-of-order writes. This ensures that the standby site can be used as the primary site when necessary without the need for resetting the standby logs. As mentioned, resetting the active logs causes data loss because the data in the active logs is ignored by the database system.

If for some reason (say due to WAN outages or high congestion) the archive log delta images cannot be shipped and the number of unarchived but the completed online redo logs reach a threshold, the primary archive replicator enters catch-up mode. The threshold can be defined as when the a spool buffer in the remote mirroring device is filled or when the archiving becomes slow enough (due to WAN delay or outages) that there is a risk of the primary database system blocking because it cannot archive the active log files. In catch-up mode, the delta images are discarded and the whole completed archive redo log images are shipped asynchronously. Of course, the primary parallel database server continues to generate archives during the network outage. However, shipping delta images to the second destination or the standby site is deferred, until the primary archive file replicator is able to ship all the pending archive redo logs to the standby site and re-enter normal mode.

On entering catch-up mode, the primary archive replicator automatically asks the remote mirroring device to pause the remote mirroring. This ensures that the online never over-shoots the archives. Including a catch-up mode allows this system to withstand WAN outages of long duration without sacrificing the operation performance of the primary site.

FIG. 4 is a flow chart illustrating a summary of the steps used to create archive files in a database system. The method for creating archive redo logs for a standby database at a

File History Content Report

The following content is missing from the original file history record obtained from the United States Patent and Trademark Office. No additional information is available.

Document Date - 2001-01-31
Document Title - Specification
Page(s) - 13

UNITED STATES PATENT AND TRADEMARK OFFICE
DOCUMENT CLASSIFICATION BARCODE SHEET



Claims

FOI b7D b7C b7E

5

Level - 2
Version 1.1

CLAIMS

What is claimed is:

- Sub
B1A1
1. A method for creating a standby data processing system located at a standby site, which is remotely located from a primary site, comprising the steps of:
- (a) computing a difference between an active data file and an archive data file to produce a delta image at the primary site;
 - (b) transmitting the active data file and the delta image, separately in time sequence, from the primary site to the standby site; and
 - (c) combining the delta image with the active data file at the standby site, to produce a standby archive data file.
2. A method as in claim 1, wherein the active data file and the archive data file are transaction logs.
3. A method as in claim 1, wherein the active data file and the archive data file are transaction redo logs.
4. A method as in claim 1, further comprising the step of storing the standby archive data file on the standby site.
5. A method as in claim 1, further comprising the step of storing transactions in the standby active data file as they are trapped at the primary site.
- Sub
B1B

02744660 "FOFET"

909

6. A method as in claim 1, further comprising the step of entering transactions from the archive data file into the standby transaction processing system.
7. A method for creating archive redo logs for a standby database at a remote location, comprising the steps of:
 - (a) trapping active log writes using a remote mirroring device that is capable of mirroring the log writes to a standby site, to create an online redo log at the standby site;
 - (b) computing a difference between an online redo log and archive redo log to create a delta image at a primary database;
 - (c) transmitting the delta image to a standby archive log replicator contained in the standby database; and
 - (d) combining the delta image with the online redo log to create a standby archive redo log on the standby database.
8. A method as in claim 7, further comprising the step of applying the standby archive redo log by creating links to the standby archive redo log.
9. A method as in claim 7, further comprising the step of transmitting the delta image to a standby archive log replicator using a first remote mirroring device in the primary database and a second remote mirroring device in the standby database.
10. A method as in claim 9, further comprising the step of using replication handles to track the progress of mirror writes on the standby database as compared to the primary database.

11. A method as in claim 10, further comprising the step of checking the primary database with database queries using the archive log replicator to determine whether archiving is complete.

12. A method as in claim 7, further comprising the step of shipping an entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously after archiving becomes slow enough that there is a risk of primary database blocking.

13. A method as in claim 12, further comprising the step of shipping an entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously when a spool buffer in a remote mirroring system has been filled.

14. A method for creating archive redo logs from a primary database, for a standby database at a remote location, comprising the steps of:

- (a) trapping file system transactions to create an online redo log at the standby database;
- (b) computing a difference between an online redo log and archive redo log in the primary database to find the delta image of the archive redo log;
- (c) transmitting the delta image to a standby database; and

09774939.013404
506
B1

(d) combining the delta image with the online redo log in the standby database to create standby archive redo log on the standby database.

15. A method as in claim 14, further comprising the step of applying the standby archive redo log by creating a link between the standby database and the standby archive redo log.

16. A method as in claim 14, further comprising the step of transmitting the delta image to a standby archive log replicator using a first remote mirroring device in the primary database and a second remote mirroring device in the standby database.

17. A method as in claim 16, further comprising the step of using replication handles to track the progress of mirror writes on the standby database as compared to the primary database.

18. A method as in claim 14, further comprising the step of checking the primary database with database queries using the archive log replicator to determine whether archiving is complete.

19. A method as in claim 14, further comprising the step of shipping an entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously after archiving becomes slow enough that there is a risk of the primary database blocking.

04/23/2018 10:04:39 AM
SUB B1

20. A method as in claim 14, further comprising the step of shipping the entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously when a spool buffer in a remote mirroring system has been filled.

21. A system for creating a standby data processing system located at a standby site, which is remotely located from a primary site, comprising the steps of:

means for computing a difference between an active data file and an archive data file to produce a delta image at the primary site;

means for transmitting the active data file and the delta image, separately in time sequence, from the primary site to the standby site; and

means for combining the delta image with the active data file at the standby site, to produce a standby archive data file.

22. An article of manufacture comprising:

a computer usable medium having computer readable program code means embodied therein for creating archive redo logs for a standby database at a remote location, the computer readable program code means in said article of manufacture comprising:

computer readable program code means for trapping active log writes using a remote mirroring device that is capable of mirroring the log writes to a standby site, to create an online redo log at the standby site;

computer readable program code means for computing a difference between an online redo log and archive redo log to create a delta image at a primary database;

computer readable program code means for transmitting the delta image to a standby archive log replicator contained in the standby database; and

SUB
B1

computer readable program code means for combining the delta image with the online redo log to create a standby archive redo log on the standby database.

00774039.013104
FOFETO" 6E5H/200

ABSTRACT

A method for creating a standby data processing system located at a standby site, which is remotely located from a primary site. The method comprises a first step of computing a difference between an active data file and an archive data file to produce a delta image at the primary site. Another step is transmitting the active data file and the delta image, separately in time sequence, from the primary site to a standby site. A final step is combining the delta image with the active data file at the standby site, to produce an archive data file. An advantage of this method is the creation of a standby data processing system, while using a lower bandwidth to create this additional site.

T9317 - Rev. 03/01
H:\Perry\Hewlett Packard\T9317\T9317_w_inventor_comments.upd

0974839-013101
T07EFO"6E84Z60

DATE RECEIVED

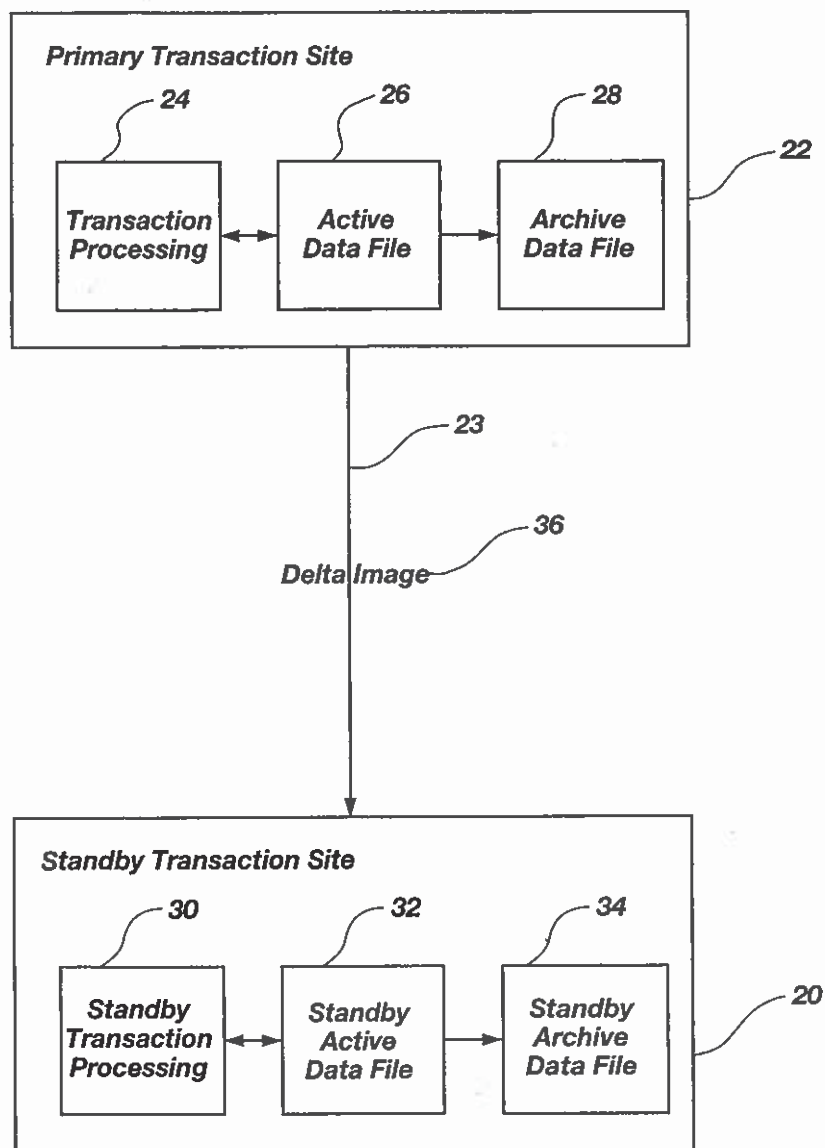


Fig. 1

09774939.0104

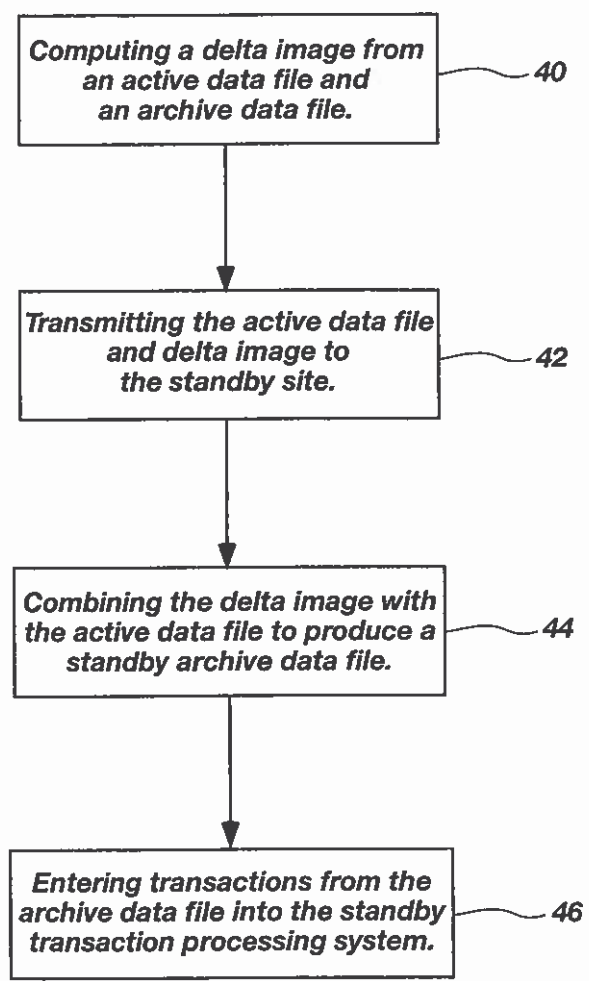


Fig. 2

FOI b6 b7C b7D

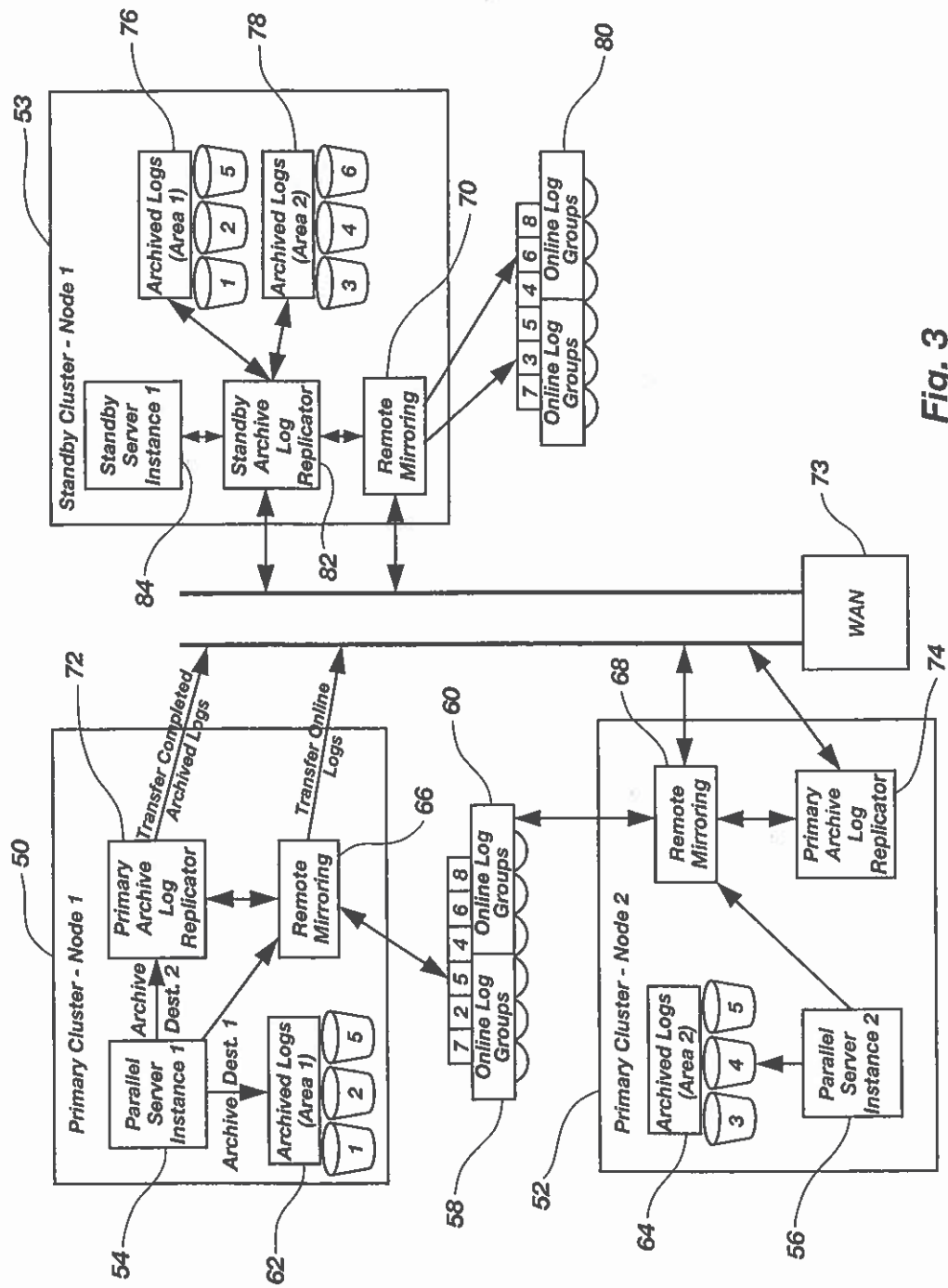


Fig. 3

TOP SECRET

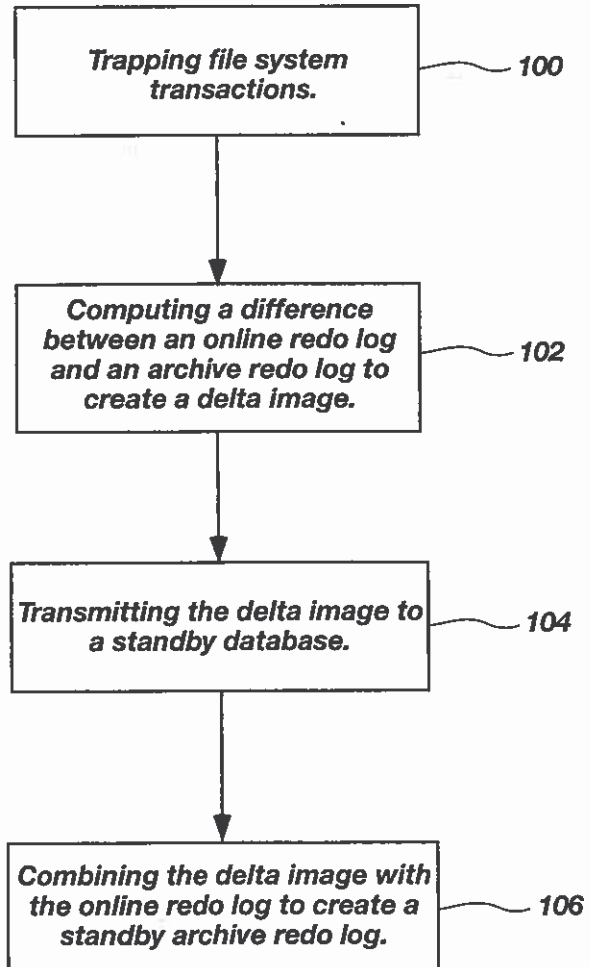


Fig. 4

IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): S. Ganesh and R. Kavanappillil

Serial No.:

Examiner:

Filing Date:

Group Art Unit:

Title: RECREATION OF ARCHIVES AT A DISASTER RECOVERY SITE



COMMISSIONER FOR PATENTS
Washington, D.C. 20231

INFORMATION DISCLOSURE STATEMENT

Sir:

This Information Disclosure Statement is submitted:

- ☒ under 37 CFR 1.97(b), or
(Within three months of filing national application; or date of entry of national application; or before mailing date of first office action on the merits; whichever occurs last)
- ☐ under 37 CFR 1.97 (c) together with either a:
☐ Statement under 37 CFR 1.97(e), or
☐ a \$240.00 fee under 37 CFR 1.17(p), or
(After the CFR 1.97 (b) time period, but before final action or notice of allowance, whichever occurs first)
- ☐ under 37 CFR 1.97 (d) together with a:
☐ Statement under 37 CFR 1.97(e), and
☐ a petition under 37 CFR 1.97(d)(2), and
☐ a \$130.00 petition fee set forth in 37 CFR 1.17(i).
(Filed after final action or notice of allowance, whichever occurs first, but before payment of the issue fee)

Please charge to Deposit Account 08-2025 the sum of \$0.00. At any time during the pendency of this application, please charge any fees required or credit any overpayment to Deposit Account 08-2025 pursuant to 37 CFR 1.25.

☒ Applicant(s) submit herewith Form PTO 1449 - Information Disclosure Citation together with copies, of patents, publications or other information of which applicant(s) are aware, which applicant(s) believe(s) may be material to the examination of this application and for which there may be a duty to disclose in accordance with 37 CFR 1.56.

☐ A concise explanation of the relevance of foreign language patents, foreign language publications and other foreign language information listed on PTO Form 1449, as presently understood by the individuals(s) designated in 37 CFR 1.56 (c) most knowledgeable about the content is given on the attached sheet, or where a foreign language patent is cited in a search report or other action by a foreign patent office in a counterpart foreign application, an English language version of the search report or action which indicates the degree of relevance found by the foreign office is listed on form PTO 1449 and is enclosed herewith.

It is requested that the information disclosed herein be made of record in this application.

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231.

Date of Deposit: 1/31/01

Typed Name: Steve M. Perry

Signature: Steve M. Perry

Rev 10/00 (IDSCERT)

Respectfully submitted,

S. Ganesh and R. Kavanappillil

By Steve M. Perry

Steve M. Perry

Attorney/Agent for Applicant(s)

Reg. No. 45,357

Date: 1/31/01

Telephone No.: (801) 566-6633

PATENT APPLICATION
Sheet 1 of 1

FORM PTO-1449 LIST OF PATENTS AND PUBLICATIONS FOR APPLICANT'S INFORMATION DISCLOSURE STATEMENT (Use several sheets if necessary)	ATTY. DOCKET NO. 10006924-1	SERIAL NO. 09/774,839
	APPLICANT Ganesh et al.	
	FILING DATE	GROUP 2172

11046 U.S. PTO
09/774839



REFERENCE DESIGNATION U.S. PATENT DOCUMENTS

EXAMINER INITIAL		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB CLASS
	1A					
	1B					
	1C					
	1D					
	1E					
	1F					
	1G					
	1H					
	1I					
	1J					
	1K					

FOREIGN PATENT DOCUMENTS

		DOCUMENT NUMBER	DATE	NAME	CLASS	SUB CLASS	TRANSLATION	
							YES	NO
	1L							
	1M							
	1N							
	1O							
	1P							

OTHER REFERENCES (including Author, Title, Date, Pertinent Pages, etc.)

1Q	Roby Sherman, "Implementing an Automated Standby Database", Oracle Magazine, May, 1999
1R	Biju Thomas, "Using Oracle8i's New Log-Analysis Tool", Oracle Magazine, September, 1999
1S	

EXAMINER [Signature]	DATE CONSIDERED 10/31/03
-----------------------------	---------------------------------

Rev 10/00 (PTO1449)

[Oracle Publishing Home](#)[Oracle Magazine](#)[Profit Magazine](#)[Current Issue](#)[Issue Archives](#)[Online Exclusive Archives](#)[Tip and Code Archives](#)

Implementing an Automated Standby Database

By Roby Sherman

Automate the maintenance of your standby database, using new features in Oracle8i

Oracle8i contains an enhanced feature that automatically keeps a standby database synchronized with a production database. This new feature, called the Automated Standby Database (ASD), greatly reduces the amount of manual work database administrators must perform to develop mechanisms for transporting archived redo logs from the production environment to the standby database.

Oracle introduced the standby database with Oracle7 as a means to clone a production database for the event of a disaster. The idea behind it is that you maintain an exact replica of your production database hardware, database, and other resources of the two environments—production and standby—should be as close to identical as possible.

As the production database operates, it creates and archives redo logs. As part of the process of maintaining a standby database, you send a copy of the archived redo logs to the standby database. There, the system applies the copies of the archived redo logs in a continuous roll-forward fashion, which keeps the two databases in sync.

If your production database experiences a catastrophic failure, you can activate the standby database as your new production environment, thereby providing near-continuous availability.

Automating the Process with Oracle8i

In releases prior to Oracle8i, the process of transporting and applying archived redo logs to the standby database was not automated unless the DBA created his or her own automated method. But now, with enhancements in Oracle8i, you can automatically and efficiently transfer and apply archived redo logs to one or more standby databases. In addition, you can take the Oracle8i ASD out of recovery mode and use it as a read-only database, allowing users to run queries against it during peak user hours.

Before you begin to establish an ASD configuration, you must answer many important questions. Although ASD can certainly be used very effectively in some production environments, there are some scenarios where ASD solutions are not appropriate. For example, if the network connection between the production and standby machines does not provide the bandwidth necessary for the timely transfer of archive logs, archive logs may begin to back up or overall network performance can be compromised.

As with any architecture change, performing research and evaluation are the keys to understanding whether or not implementing a standby database is appropriate for your particular environment.

Setting Up An Automated Standby Database

The first step in creating a standby environment is to configure the standby server machine to resemble the production machine's configuration as closely as possible. Although it is not an absolute requirement, the more similar the standby's configuration mirrors the production environment, the better, to maximize operational smoothness and consistency. This is especially important if you plan to operate your databases by having the production database and the standby database swap roles.

failure occurs.

The following steps refer to the servers listed in Table 1. The production instance, dtmprod1, is currently running in ARCHIVELOG mode and resides on server rs-co1. The standby database will be called and reside on the server rs-co2, which has been configured similarly to the production instance.

Oracle recommends that the file-system and raw-device layouts remain identical on both machines, have made them slightly different in order to illustrate the use of the init.ora parameters DB_FILE_NAME_CONVERT and LOG_FILE_NAME_CONVERT in Oracle8i.

Step 1. Evaluate Your Environment. Make sure you have assessed your environment by answering basic questions before you proceed to enable ASD:

- *How often should a log switch occur?* If your redo logs are too small and you are getting frequent switches, you may not have sufficient time to resynchronize your standby environment manually before new archive logs are generated. For our particular environment, we sized the dtmprod production database redo logs at 20MB each, which caused the database to switch once every minute, providing enough time to allow the archive log transfer to complete and also allowing for a manual resynchronization if necessary. Naturally, these values will be specific to your environment.
- *Approximately how many transactions are in each redo-log file?* If you increase the size of your logs to reduce the number of log switches, you will be storing more transactions in each file. This means that the amount of time required to ship the archived redo log to the remote side will increase, potentially causing a network bottleneck. The number of transactions per log will also be greater, increasing the amount of potential data loss if the production failure occurs before the system can transmit all archived redo logs. So, for example, in our case, we didn't know the number of transactions, but we did know that our DML changes were committed almost immediately and a network could transfer the archive logs from the production database to the standby database in 5 minutes. Based on this information, we estimated that in a worst-case scenario, we would have 30 minutes' worth of transactions.
- *How will the system forward the application users seamlessly to the standby should the production database fail?* When the production database fails, the users need a way to connect to the standby if it is activated. Depending on your environment, you can create one of many Net8-based solutions (Net8 is the enhanced version of SQL*Net that ships with Oracle8), using Net8's failover facility you can create coding-specific solutions to this problem. The solutions have different advantages and disadvantages, ranging in complexity from a simple tnsnames.ora modification on the client to server IP/DNS reconfigurations. In our situation, to be able to provide near-continuous data availability, we developed an application-routing API that redirects inbound user requests to the standby database. We can easily instruct the API to redirect inbound requests to the standby database, if it has been activated.
- *Is there a sufficient number of logfile groups?* If you have only a small number of logfile groups, the ARCH process may not have enough time to archive the inactive log to the standby before the production database requires the file for reuse. If this occurs, the database will hang momentarily until archiving completes, assuming that your production database is configured to treat the standby as a mandatory destination.

Step 2. Validate Net8 Connectivity. The production database dtmprod1 will communicate with dtmback1 via Net8. This means that you must set up the listener on the rs-co2 standby system to accept connections from the database rs-co1. You may also need to place a tnsnames.ora entry for dtmback1 on the rs-co1 production system if you are not using Oracle Names in your environment. Use SQL*Plus on the production database to verify that you have configured connectivity properly and that the listener is functioning properly on the remote server.

Step 3. Back up the Production Database. Next, prepare the production database on rs-co1 for transport to the standby on rs-co2. If user activity might occur during this process, you should place the database in a state that allows for transport.

restricted mode prior to starting these procedures, to prevent the introduction of changes into the database. Perform the following actions to archive the current online redo log and to create a standby database file, using a SYSDBA account such as SYS or SYSTEM with SQL*Plus, svrmgrl, or another comparable utility:

```
SVRMGR> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

```
SVRMGR> ALTER DATABASE CREATE STANDBY CONTROLFILE
```

```
AS '/pathname/filename.ctl'
```

Next, cleanly shut down your database. Then copy the production database's datafiles to equivalent locations on the standby database host. Do not change the names of these files when moving them between different platforms. Make sure you copy

- All tablespace data files
- All redo logs
- All archived redo logs currently online
- The standby control file created in the previous step (be sure to place copies of this file with the appropriate name in the equivalent locations on the standby where the production control file have existed, if you had copied them)

However, make sure you do not copy

- The production control files (the standby will use the standby control file you generated previously)
- The production init.ora files (you will copy these later)

Step 4. Modify the init.ora File on the Production Database. Oracle8i includes new and modified parameters for automating the standby-database maintenance process (see Table 2 for a list of the new init.ora parameters). In our example, the production database, dtmprod1, is configured to archive to a singular file-system destination, using the log_archive_dest parameter. You must modify it as shown in Listing 1 to automatically ship the archived redo logs to the remote standby.

Step 5. Copy the Production init.ora File(s) to the Standby Database for Modification. Next, copy the production init.ora file to the appropriate directory on the standby database server. Once you have copied the file, open it and make these additional changes on the standby server, as shown in Listing 2:

- Update the CONTROL_FILES parameter so that it points to all of the standby control files on the standby database server.
- Add the STANDBY_ARCHIVE_DEST parameter and set its value to the location where the standby database should store any archived redo logs received from the production database.
- Make any applicable changes to the LOG_ARCHIVE_DEST_n parameters (or comment them out) to point to appropriate log-archive storage locations to be employed if the standby database is a

Step 6. Start the Standby Database. The standby database is now ready to be put into service. To perform the following actions on it, using a SYSDBA account such as SYS or SYSTEM with SQL*Plus, svrmgrl, or another comparable utility:

```
SVRMGR> CONNECT SYS AS SYSDBA
```

```
SVRMGR> STARTUP NOMOUNT
```

```
SVRMGR> ALTER DATABASE MOUNT STANDBY DATABASE;
```

```
SVRMGR> ALTER DATABASE RECOVER MANAGED
```

```
STANDBY DATABASE;
```

or

```
SVRMGR> ALTER DATABASE RECOVER MANAGED STANDBY
```

```
DATABASE TIMEOUT nn;
```

(nn = the number of minutes to wait for an archived redo log before recovery is automatically c

In Oracle8 Release 8.1.5, the ALTER DATABASE RECOVER MANAGED STANDBY DATABASE command does not return until it is canceled (from another session) or you press CTRL-C. I recommend you execute this command on the console for the standby server so that it can remain open indefinitely.

Step 7. Restart the Production Database. Now that the standby database is ready to receive archived redo logs, you are ready to restart the production database. Perform the following actions on the production database using a SYSDBA account such as SYS or SYSTEM with SQL*Plus, svrmgrl, or another comparable utility.

```
SVRMGR> CONNECT SYS AS SYSDBA
```

```
SVRMGR> STARTUP
```

Step 8. Verify Production to Standby Transfer Operations. Finally, validate that the production database is able to automatically transmit logs to the standby database and that the standby is ready to receive archived redo logs. To make sure that the production database is transmitting the logs, execute the following command on the production database:

```
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

When sufficient time has passed for the production database to transport the archived redo log, perform a query on the production database:

```
SQL select * from v$archive_dest;
```

Find the row in the output that corresponds to your standby database, and verify that the values for `BINDING`, `REOPEN_SECS`, and `DESTINATION` are correct, based on your requirements. Make sure the value for `STATUS` is `VALID`—if not, check the `FAIL_DATE`, `FAIL_SEQ`, and `ERROR` column error.

Next, verify that the directory specified in `standby_archive_dest` `init.ora` parameter on the standby contains the appropriate archived redo log that should have been copied from production. Make a note of the log's sequence number.

Check the last entries of the standby's alert log for information about the log received by the standby database. Verify that the log-sequence number for which the standby was waiting matches the sequence number of the actual archived log in the `standby_archive_dest` directory (the entry appears as "Media Recovery for thread n seq# xyz"). Finally, make sure that Media Recovery applied the log (the entry appears similar to "Media Recovery Log /pathname/logxyz_n.arc"). See [Listing 3](#).

With Oracle8 Release 8.1.5, if the above conditions have not been met, you must cancel the standby automatic recovery and manually transport and apply all missing archived redo logs.

Once you have applied the logs and restarted the automated standby, the log-sequence number between the standby database should match the current log sequence in the production database.

Activating the Standby Database

You now have an operational, automatically updated standby database that you may activate in the catastrophic production failure. Once activated, the standby database becomes a production-like instance and cannot be converted back to its previous form. If the standby database is already in `ASD` mode when the failure occurs, follow these steps to activate the standby database:

1. If possible, archive your current production database logs by issuing `ALTER SYSTEM ARCHIVE LOG CURRENT` on the production database. Copy the current online redo log—along with remaining archived logs that have not yet been transported—to the standby database server.
2. If your standby database has not timed out from your recovery, simply open a new SQL session on the standby database, by using a `DBA` account, and issue

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY
```

```
DATABASE CANCEL;
```

3. Locate the end of the standby database's alert.log, and identify the last archived log that was applied.
4. Manually apply any remaining logs to the standby database:

```
SQL> ALTER DATABASE RECOVER [FROM 'pathname']
```

```
STANDBY DATABASE;
```

5. When you have applied the remaining logs to the standby database, stop the recovery by issuing

```
SQL> ALTER DATABASE RECOVER CANCEL;
```

6. Convert the standby database to a production environment:

```
SQL> ALTER DATABASE ACTIVATE STANDBY DATABASE;
```

7. Shut down the standby database. The standby is now a full-fledged production database. As the standby database is the only working database at this point, it is highly advisable that you immediately perform a backup before starting the database for production use in the event of emergency.
8. Start the new production database as you would start any other database.
9. Initiate any and all necessary processes that will redirect users smoothly to the new production database.

Now that your database users have safely migrated to the standby database, you can begin the process of recovering your production database.

ASD Maintenance Tips

- Although the REOPEN parameter will attempt to reestablish the connection to a failed standby destination, it will (re)transmit the missed archive logs only if the destination is specified as MANDATORY. If a destination is OPTIONAL, the REUSE parameter will not attempt to transmit a log that hasn't been sent. You must manually transfer and apply these logs, plus any logs that transmitted since the REOPEN was successful, before ASD can resume. Check the production database alert.log to see if a log sequence failed to transmit to the standby database.
- Actions such as altering the physical structure, performing direct loads, and executing unrecoverable operations against your production environment can have serious impacts on the standby database. Although these actions are too numerous for the scope of this article, it is critical that you understand them. For more information, see the *Oracle8 Backup and Recovery Handbook*.
- Oracle8i provides the facility to open a standby database in read-only mode and perform the operations of a temporary reporting database. Although this functionality may be useful for your environment, be aware that additional logs transmitted from production to the standby while the standby is operating as a reporting database will not be applied until the standby is placed back into ASD recovery mode, which can increase the amount of downtime in the event of a production failover. If you want to take advantage of the ASD and read-only features, it may be more appropriate to create two standby databases.

Conclusion

Regardless of how extensively you plan for emergencies, disasters beyond your control can still happen. Disasters that can compromise your database and possibly bring your business to a halt. In many such environments, you can use Oracle8i's new Automated Standby Database functionality to reduce production database outages, helping your database users continue to work.

Oracle's ASD lets you create and automatically maintain a type of spare-database facility. When you implement such a configuration, the standby database automatically applies archived logs from the primary database. Then, if an outage occurs that removes the primary database from service, the standby database is activated to serve as the new primary database for the business, with minimal loss of time or data.

Roby Sherman (rxsherm@usa.net) has worked with Oracle databases for more than eight years. As Oracle database administrator and architect for the Data Technology Management group at U S Communications, his charter is to evaluate emerging Oracle database technologies and guide the implementation of applied architectures that exploit this functionality properly.

TABLE 1: Production and standby file systems on servers rs-co1 and rs-co2

Server rs-co1	Server rs-co2	Description
/opt/dbms1/app/oracle /product/8.1.5	/opt/dbms1/app/oracle /product/8.1.5	\$ORACLE
/opt/dbms1/app/oracle /admin/dtmprod1	/opt/dbms1/app/oracle /admin/dtmback1	Contains di PFIL, BD etc.
/opt/vol01/oradata /dtmprod1	/opt/vol01/oradata /dtmback1	Datafiles f databases
/opt/vol02/oradata /dtmprod1	/opt/vol02/oradata /dtmback1	Datafiles f databases
/opt/vol03/oradata /dtmprod1	/opt/vol03/oradata /dtmback1	Datafiles f databases
/opt/vol04/oradata /dtmprod1	/opt/vol04/oradata /dtmback1	Datafiles f databases

TABLE 2: init.ora parameters in Oracle8i for automating standby-database maintenance.

LOG_ARCHIVE-DEST_n:	Defines from one to five destinations and attributes for archived redo logs. These destinations can be operating-system locations (file system directories, other devices, etc.) or the names of standby databases accessible via Net8.
LOG_ARCHIVE_DEST_STATE_n:	Specifies which LOG_ARCHIVE destinations are enabled or excluded from archiving operations.
LOG_ARCHIVE_MAX_PROCESSES:	Specifies the maximum number (up to 10) of ARCH processes for the database. Each ARCH process is responsible for writing a single inactive log to the multiple destinations specified in init.ora.

LOG_ARCHIVE_MIN_SUCCEED_DEST:	If LOG_ARCHIVE_DEST_n destinations are not specified as MANDATORY, this optional parameter can be used to indicate the number of filesystems/local destinations that MUST succeed in order to mark the online redo log ready for reuse.
STANDBY_ARCHIVE_DEST:	Specifies the file-system location where the standby database will write archived redo logs transferred from the production database.
DB_FILE_NAME_CONVERT:	Used to convert string patterns.
LOG_FILE_NAME_CONVERT:	Used to convert production values to standby values.

[Oracle Publishing Home](#)[Oracle Magazine](#)[Profit Magazine](#)[Current Issue](#)[Issue Archives](#)[Online Exclusive Archives](#)[Tip and Code Archives](#)

Practicing Effective Redo-Log Placement

By James Koopmann

Monitoring your archiving process can help you identify bottlenecks and optimize your system performance.

How much do you know about your archiving process? Although archiving generally does not negatively affect database performance, an archive process that depends on poorly placed or inadequately sized online redo logs could result in a system bottleneck.

Naturally, the solution to a sluggish archiving process is to increase the number of online redo logs or to enlarge the existing redo logs. But where do you get relevant information about your archiving process to help you make these decisions?

There are a few different ways you can look at the archiving process to determine whether it is performing optimally. Go through the procedures detailed in this article to get a good idea of how your archiving process performs. Then, tweak the `init.ora` parameters to optimize archiving performance.

I won't cover the basics of archiving in this article. Nor will I discuss the `init.ora` parameters that you use to tune your archiving process. Many good manuals on these topics exist. Rather, my method identifies the impact your tuning has on the system. The script output will help you determine if the tweaking you have done has given you the results you want. I recommend performing the procedures in this article before and after you make changes to the `init.ora` parameters.

Also, practice good system maintenance and make sure you document your system before making changes to it, in case you must undo any modifications.

Locating the Logs

The first files to review when looking into the archive process are the data files associated with the redo logs. Remember that the archive process reads from one group while it is writing to another. Poorly planned placement of these files or any other data-file access—Oracle or non-Oracle—is detrimental to the archiving process. It's important that these files be placed on devices used for little other than redo activity.

You need to find out where the database thinks the online redo logs are, as opposed to where the logs are really located. Running the simple query in [Listing 1](#) will show where the database thinks the logs are.

After you get this list, you should verify that these files and directories don't have any symbolic links, indicating that the logs are actually somewhere else on disk. You do the verification by using a command as simple as `"ls -l"` on the directories and the redo-log members.

After you've verified their location, put the redo logs on their own disks without any other highly accessed data files.

Next, use the script (logactivity.sql) in [Listing 2](#) to check for things such as the number of groups, number of members per group, size of group members, status of the group, and time between log switches.

The output of the logactivity.sql script shows that this particular instance has nine redo-log groups with two members each, and each member is 20 megabytes in size. Another key point is the order in which the groups are used. The report shows that the groups are used in a cyclical manner (1, 2, 3 . . .) instead of in random order. This is important information to have when you evaluate the placement of the data files in relation to the order in which they are written. For example, if the files are written cyclically, it would be inefficient to have groups 1 and 2 on the same disk, because the archive process would read from group 1 to write the archive logs while a new redo was written to group 2.

The report also includes a FIRST_TIME column, which shows when the first system-change number (SCN) was written to the group in question. You use this number to determine how long Oracle wrote to a specific log member. For example, in the FIRST_TIME column in [Listing 2](#), you can see that Oracle wrote to sequence number 1697 for about seven minutes.

The alert.log report, shown in [Listing 3](#), displays the information logged for this system when archive logs are switched. This report shows when the log switch began, the log sequence number used, and the current online log we will be using.

If you look at the output shown in [Listing 2](#) and [3](#), the FIRST_TIME value in [Listing 2](#) matches almost exactly the time stamp for the corresponding sequence number in the alert.log in [3](#). This indicates the start time of the archive process for the previous online redo log.

In addition, the time stamp on the physical archived log in [Listing 4](#) shows the ending time of the archive process. If you subtract the start time from the ending time for a specific report, the result is the time it takes to write out an online redo log to archive. In this example, the average time to cut an archive log is anywhere from 60 seconds to 90 seconds.

Now that you've figured out how long it takes to write a physical archive log to disk, you should next determine how many logs you are cutting every hour. To do this, look in the alert.log and count them or employ the logdist .sql script in [Listing 5](#) to produce a tabular picture of your log-cutting history.

Can Your Disk Manage?

Now that you have determined that logs are being produced and where they are stored on disk, you must find out how well the disk placement is holding up to the demands of the archiving process.

In order to perform this next step, you need to know the actual disks and devices where the redo logs and archives are stored. (You may need to consult your system administrator for this information.)

For example, my system has the disk-to-device mappings shown in [Listing 6](#). The placement of online redo logs on disk discovered in [Listing 1](#) and the order in which they are created and used ([Listing 2](#)) is important in order to reduce resource contention between redo-log members.

When you create your log groups and members, start with group 1 and continue creating them sequentially for the number of groups you need. Of equal importance is the order in which you create the members in each group: Always create member 1 before member 2. Oracle will try to use the first member in a group when creating the archive log for the group. Therefore, if you have to do maintenance on the members or groups, you may have to drop and re-create other groups in order to maintain the correct order.

[Listing 7](#) shows a section of an operating-level iostat. In this case, my system was writing to redo-log group 3. When group 3 became full, group 3 was archived and writes went to group 4. In the first five lines, all writing is going to sd9 (redo0301) and sd50 (redo0302). But in the last five lines, we read only from sd9 (redo0301) for archiving to sd17 (/dbarch) and continue writing the next group to sd11 (redo0401) and sd50 (redo0402). This makes it so that no single disk must handle more than one I/O operation for more than one redo-log group.

Of course, if you cut more redo than the archive process can handle, you may have to either spread your redo logs over more disks or put up with some resource contention between the redo-log groups.

After using the scripts I've written to analyze your archiving process, you can tweak the init.ora parameters to your heart's content. Without using these scripts, it might have been difficult to determine whether or not those increases or decreases actually helped the underlying archiving processes for your system. But using these methods to examine your archive process more closely should help you determine whether or not your changes have, in fact, helped your system.

James F. Koopmann (james@infobeat.com) is a database administrator for Infobeat, Inc. in Denver, Colorado. Visit his Web site at www.pcisys.net/~jkoopmann.

Using Oracle8i's New Log-Analysis Tool

By Biju Thomas

Audit database use and maintain data integrity with Oracle8i's LogMiner.

Oracle redo log files contain a wealth of information on the activities in an Oracle database. Log files store all the data needed to perform a database recovery and record every change made to the database structure. But until Oracle8i, there was no built-in tool for helping administrators read and interpret the redo-log files.

Oracle8i provides a powerful new tool, LogMiner, that enables administrators to audit database use and to undo erroneous data changes without performing a full recovery. LogMiner enables small as well as large enterprises to fully understand activities taking place in the database. It is a complete relational tool that allows DBAs to read log files and use SQL to analyze and interpret the information contained in them. The fact that LogMiner can view any valid redo-log file—online or archived—(from Oracle8 forward) means that it imposes no additional data-collection overhead, because the Oracle system already maintains the redo-log files for database recovery.

LogMiner is actually a set of PL/SQL packages and dynamic (V\$) views. The PL/SQL packages that comprise LogMiner are already installed, along with the Oracle8i database, as part of the catproc.sql script that installs the built-in Oracle packages.

LogMiner is useful for

- *Tracking changes to the database:* Using LogMiner, you can track changes to the database offline, without affecting the online performance of the database.
- *Undoing changes to the database:* You can undo specific changes to data, minimizing the need for point-in-time recovery.
- *Tuning and capacity planning:* Using LogMiner, you can analyze data in the redo logs and study data-growth patterns.

Preparing to Use LogMiner

LogMiner is composed of two new PL/SQL built-in packages (DBMS_LOGMNR and DBMS_LOGMNR_D) and four V\$ views. (The V\$ views are created when you start LogMiner by using the DBMS_LOGMNR.START_LOGMNR procedure.) You'll use the DBMS_LOGMNR_D package first to export the data dictionary (from the database you want to analyze) as a text file for use with LogMiner. This dictionary file is optional, but you won't be able to easily comprehend LogMiner results without it, because column headings and the like will be presented as hex values. For example, instead of the SQL statement

```
INSERT INTO emp (name, salary)
VALUES ('John Doe', 50000);
```

LogMiner will display

```
insert into Object#2581(col#1, col#2)
values (hextoraw('4a6f686e20446f65'), hextoraw('c306'));
```

Follow the steps in the sidebar "Creating a Dictionary File" to create a dictionary file.

Analyzing Your Logs

Once you have created a dictionary file, you can begin analyzing redo logs.

- **Step 1.** Use the `ADD_LOGFILE` procedure to specify the log files you want to analyze. You can use the `DBMS_LOGMNR.START_LOGMNR` procedure to specify the exact time or system change number (SCN).
- **Step 2.** Create a list of logs by specifying the `NEW` option in the `DBMS_LOGMNR.ADD_LOGFILE` procedure. The following specifies `/oracle/logs/log1.f`:

```
execute dbms_logmnr.add_logfile(
LogFileName => '/oracle/logs/log1.f',
Options => dbms_logmnr.NEW);
```

If you desire, add more logs with the `ADDFILE` option. The following adds `/oracle/logs/log2.f`:

```
execute dbms_logmnr.add_logfile(
LogFileName => '/oracle/logs/log2.f',
Options => dbms_logmnr.ADDFILE);
```

To remove logs, use the `REMOVEFILE` option. The following removes `/oracle/logs/log2.f`:

```
execute dbms_logmnr.add_logfile(
LogFileName => '/oracle/logs/log2.f',
Options => dbms_logmnr.REMOVEFILE);
```

Now, you can start LogMiner and begin your analysis.

- **Step 3.** Issue the `DBMS_LOGMNR.START_LOGMNR` procedure to start the LogMiner utility. To start LogMiner, using the dictionary file located in `/oracle/dictionary.ora`, issue the following:

```
execute dbms_logmnr.start_logmnr(
DictFileName => '/oracle/dictionary.ora');
```

To narrow the scope of the log or logs you want to analyze, you can set several different parameters in the `DBMS_LOGMNR.START_LOGMNR` procedure (see Table 1). You can set the `StartTime` and `EndTime` parameters to filter data by time. The procedure expects date values, so you will have to use the `TO_DATE` function to specify the date and time:

```
execute dbms_logmnr.start_logmnr(
DictFileName => '/oracle/dictionary.ora',
StartTime => to_date('01-Jan-98 08:30:00',
```

```
'DD-MON-YYYY HH:MI:SS')
EndTime => to_date('01-Jan-1998 08:45:00',
'DD-MON-YYYY HH:MI:SS'));
```

You can use the StartScn and EndScn parameters to filter data by the SCN:

```
execute dbms_logmnr.start_logmnr(
DictFileName => '/oracle/dictionary.ora',
StartScn => 100,
EndScn => 150);
```

DBMS_LOGMNR.END_LOGMNR ends the LogMiner session.

Analyzing Archived Redo-Log Files from Other Databases

You can take advantage of LogMiner to analyze redo-log files that were generated from database instances other than the one along with which LogMiner is installed. To use LogMiner to analyze logs from another database, be sure to keep the following in mind:

- LogMiner must be able to access a data-dictionary file that is created from the same database as the redo-log files and that uses the same database character set.
- You must run LogMiner on the same platform as the one that generated the log files, although it does not need to be the same hardware. So if you want to use LogMiner to examine files that were generated by an Oracle8i database running on Microsoft Windows NT Server, you must also be running LogMiner on an Oracle instance running on Windows NT Server, and not, say, Solaris or Linux.
- You must use redo-log files that can be applied for recovery from Oracle8 and later releases; you cannot use redo-log files from Oracle7 or Oracle6 database instances.

LogMiner in Action

There are many ways in which you can use LogMiner in a normal business day. For example, consider the situation in which a database user—Joe—performs an activity that must later be reversed, and you know only that the activity occurred between 8:45 a.m. and 9:30 a.m.

In this case, the dictionary file is dict.ora, and it is stored under /user/local/data. Make sure the init.ora parameter UTL_FILE_DIR is set to /user/local/data. Then, start up the database and create the dictionary, using:

```
Execute dbms_logmnr_d.build (
Dictionary_filename => 'dict.ora',
Dictionary_location => '/user/local/data');
```

Now, add to the list the names of the log files created during the time period. For the sake of the example, assume that two log files—arch_234.log and arch_235.log—were created. Create the new list and then add the second file to it, as follows:

```
Execute dbms_logmnr.add_logfile
( Options => dbms_logmnr.new,
Logfilename => '/arch/PROD/arch_234.log');
```

```
Execute dbms_logmnr.add_logfile
      ( Options => dbms_logmnr.addfile,
    Logfilename => '/arch/PROD/arch_235.log');
```

Next, start LogMiner using the dictionary file you created above, and limit the search to the specific time range during which Joe made his changes to the database:

```
execute dbms_logmnr.start_logmnr(
  DictFileName => '/user/local/data/dict.ora',
  StartTime => to_date('15-Mar-1999 08:45:00',
    'DD-MON-YYYY HH:MI:SS')
  EndTime => to_date('15-Mar-1999 09:30:00',
    'DD-MON-YYYY HH:MI:SS'));
```

Now, the V\$LOGMNR views are populated with the database manipulation language (DML) statements applied to the database during the time frame you specified. Only the session that performed the analysis to view the log information can query the views.

This simple example shows how powerful and easy to use LogMiner is:

```
SELECT sql_redo, sql_undo
FROM   v$logmnr_contents
WHERE  username = 'JOE'
AND    tablename = 'SALARY';
```

SQL_REDO

```
-----
delete * from SALARY
where EMPNO = 12345
insert into SALARY(NAME, EMPNO, SAL)
values('JOEDEVO', 12345, 2500)
2 rows selected
```

SQL_UNDO

```
-----
insert into      SALARY(NAME,EMPNO, SAL)
values ('JOEDEVO', 12345,500) and ROWID = 'AAABOOAABAAEPCABA';
delete * from SALARY
where EMPNO = 12345 and ROWID = 'AAABOOAABAAEPCABA';
```

With this information, you can undo the operation—in this case, return Joe to his previous salary.

More Information

As a DBA, you can use LogMiner to get a wealth of information about database activities. One of the most valuable uses of LogMiner is to help recover or undo certain changes without having to do a full database recovery. In addition, the tool is very useful for monitoring or auditing user activity. For example, you can use LogMiner to discover who modified the data and what the data looked like before it was modified. You can analyze redo-log files generated by any Oracle8 or later database. An especially convenient feature of LogMiner is its ability to analyze log files from databases other than the one in which it is installed.

Visit the Oracle Technology Network (OTN; <http://technet.oracle.com>) for complete Oracle8i and LogMiner reference documentation. Membership in OTN

is free and gives you online access to many useful resources.

Biju Thomas (tbiju@hotmail.com) is an Oracle certified database administrator at Renaissance Worldwide Inc., in Fort Worth, Texas. He has more than six years' experience with Oracle databases and development tools and has written many tips for the IOUG-A Technical Repository. See his introduction to Oracle database administration at <http://welcome.to/OracleDBA59dba2.html>.

TABLE 1: The parameters of the DBMS_LOGMNR.START_LOGMNR procedure.

StartScn	IN NUMBER default 0	Use the redo logs that have a SCN greater than or equal to the number you specify.
EndScn	IN NUMBER default 0	Use the redo logs that have a SCN less than or equal to the number you specify.
StartTime	IN DATE default '01-jan-1998'	Use the redo logs that have a time stamp greater than or equal to the time you specify.
EndTime	IN DATE default '01-jan-2899'	Use the redo logs that have a time stamp less than or equal to the time you specify.
DictFileName	IN VARCHAR2 default "	Include the file containing a snapshot of the database catalog. Using this file means the SQL_REDO and SQL_UNDO columns will be included in V\$LOGMNR_CONTENTS as well as in the fully translated SEG_NAME, SEG_OWNER, SEG_TYPE_NAME, and TABLE_SPACE columns.
Options	IN BINARY_INTEGER default 0	Indicate if a column map is specified in the logmnr.opt file. This file must be in the same directory as the dictionary file specified by DictFileName. The V\$LOGMNR_CONTENTS table includes multiple sets of placeholder columns. Each placeholder column set contains a name column, a redo-value column, and an undo-value column. Each placeholder

		column. Each placeholder column set can be assigned to a table and a column via an optional LogMiner assignment file (logmnr.opt). After a placeholder column has been assigned, it can be used to select changes to the assigned column and table from the redo log stream.
--	--	--

Creating a Dictionary File

Create a dictionary file by mounting a database, extracting dictionary information, and saving it to an external file. You must create the dictionary file from the same database that generated the log files you want to analyze. Once you have created the dictionary file, you can use it in conjunction with LogMiner to analyze redo logs.

Create an Oracle8i Database Dictionary File

- **Step 1.** Specify the directory to which you want DBMS_LOGMNR_D to export your data dictionary as a text file by setting the UTL_FILE_DIR parameter in the server's init.ora file. For example, the following sets it to use /oracle/logs:

```
UTL_FILE_DIR = /oracle/logs
```

- **Step 2.** Use Server Manager to mount and then open the database that contains the files you want to analyze:

```
SVRMGR> startup
```

- **Step 3.** Execute the PL/SQL procedure DBMS_LOGMNR_D.BUILD. Specify a filename for the dictionary and a directory pathname for the file. This procedure creates the dictionary file, which you should use to analyze log files. For example, the following creates the file dictionary.ora in /oracle/logs:

```
EXECUTE dbms_logmnr_d.build(
dictionary_filename => 'dictionary.ora',
dictionary_location => '/oracle/logs');
```

Creating an Oracle8 Database Dictionary File

Although LogMiner is new with the release of Oracle8i, you can also use the tool to analyze redo logs from Oracle8 Release 8.0 databases.

- **Step 1.** Use an operating-system utility to copy the dbmslogmnr.sql script—which is in the \$ORACLE_HOME/rdbms/admin directory on the Oracle8i server—to the same directory on the Oracle8 server. For example, enter the following:

```
% cp /8.1/rdbms/admin/dbmslogmnr.sql  
    /8.0/rdbms/admin/dbmslogmnr.sql
```

- **Step 2.** Use Server Manager to mount and then open the database that contains the files you want to analyze:

```
SVRMGR> startup
```

- **Step 3.** Execute the copied dbmslogmnr.sql script on the 8.0 database to create the DBMS_LOGMNR_D package.
- **Step 4.** Specify a directory for the PL/SQL package to use by setting the init.ora parameter UTL_FILE_DIR. If you do not reference this parameter, the procedure will fail. For example, the following sets the init.ora parameter to use /8.0/oracle/logs:

```
UTL_FILE_DIR = /8.0/oracle/logs
```

- **Step 5.** Execute the PL/SQL procedure DBMS_LOGMNR_D.BUILD. Specify both a filename for the dictionary and a directory pathname for the file. This procedure creates the dictionary file, which you should use to analyze log files. For example, enter the following to create file dictionary.ora in /8.0/oracle/logs:

```
EXECUTE dbms_logmnr_d.build(  
dictionary_filename => 'dictionary.ora',  
dictionary_location => '/8.0/oracle/logs');
```

Copyright © 1999 Oracle Corporation. All Rights Reserved.

File History Content Report

The following content is missing from the original file history record obtained from the United States Patent and Trademark Office. No additional information is available.

Document Date - 2001-03-13
Document Title - USPTO Communication Re: Change of Address



UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
UNITED STATES PATENT AND TRADEMARK OFFICE
WASHINGTON, D.C. 20231
www.uspto.gov

APPLICATION NUMBER	FILING/RECEIPT DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NUMBER
09/774,839	01/31/2001	Shivaji Ganesh	10006924-1

CONFIRMATION NO. 6266

FORMALITIES LETTER



HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, CO 80527-2400

Date Mailed: 03/14/2001

NOTICE TO FILE MISSING PARTS OF NONPROVISIONAL APPLICATION

FILED UNDER 37 CFR 1.53(b)

Filing Date Granted

An application number and filing date have been accorded to this application. The item(s) indicated below, however, are missing. Applicant is given TWO MONTHS from the date of this Notice within which to file all required items and pay any fees required below to avoid abandonment. Extensions of time may be obtained by filing a petition accompanied by the extension fee under the provisions of 37 CFR 1.136(a).

- The oath or declaration is missing.
- To avoid abandonment, a late filing fee or oath or declaration surcharge as set forth in 37 CFR 1.16(e) of \$130 for a non-small entity, must be submitted with the missing items identified in this letter.
- The balance due by applicant is \$ 130.

*A copy of this notice **MUST** be returned with the reply.*

Sally
Customer Service Center
Initial Patent Examination Division (703) 308-1202
PART 3 - OFFICE COPY

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P. O. Box 272400
Fort Collins, Colorado 80527-2400

PATENT APPLICATION

ATTORNEY DOCKET NO. 10006924-1



IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

Shivaji Ganesh et al

Serial No.: 09/774839

Examiner: N/A

Filing Date: Jan 31, 2001

Group Art Unit: N/A

Title: Recreation Of Archives At A Disaster Recovery Site

COMMISSIONER FOR PATENTS
Washington, D.C. 20231

TRANSMITTAL LETTER FOR SUBMITTAL OF MISSING PARTS

Sir:

This is in response to a Notice to File Missing Parts of Application under 37 CFR 1.53(f) mailed on 3/14/01. Enclosed is a copy of said Notice and the following documents and fees to complete the filing requirements of the above-identified application.

(X) Executed Declaration and Power of Attorney. The above-identified application is the same application which the inventor executed by signing the enclosed declaration.

() Statutory basic filing fee () Utility \$710.00 () Design \$320.00

() Additional claim fees of \$

(X) Missing Parts Surcharge \$130.00

() A Petition for Extension of Time for reply to Notice of Missing Parts is attached.

() one month \$110.00
() two months \$390.00
() three months \$890.00
() four months \$1390.00

Please charge to Deposit Account 08-2025 the sum of \$130.00. At any time during the pendency of this application, please charge any fees required or credit any overpayment to Deposit Account 08-2025 pursuant to 37 CFR 1.25.

(X) A duplicate copy of this transmittal letter is enclosed.

Respectfully submitted,

Shivaji Ganesh et al

By T. Grant Ritz

T. Grant Ritz

Attorney/Agent for Applicant(s)
Reg. No. 39,819

Date: 5-2-01

Telephone No.: (970) 898-0697

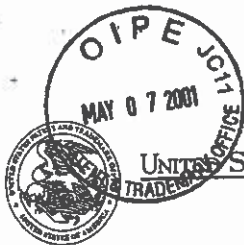
I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231.

Date of Deposit: 5-3-01

Typed Name: Cathi Christensen

Signature: Cathi Christensen

Rev 10/00 (MisParts)



UNITED STATES PATENT AND TRADEMARK OFFICE

COMMISSIONER FOR PATENTS
UNITED STATES PATENT AND TRADEMARK OFFICE
WASHINGTON, D.C. 20231
www.uspto.gov

APPLICATION NUMBER	FILING/RECEIPT DATE	FIRST NAMED APPLICANT	ATTORNEY DOCKET NUMBER
09/774,839	01/31/2001	Shivaji Ganesh	10006924-1

CONFIRMATION NO. 6266

FORMALITIES LETTER



OC000000005861817

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, CO 80527-2400

Date Mailed: 03/14/2001

NOTICE TO FILE MISSING PARTS OF NONPROVISIONAL APPLICATION

FILED UNDER 37 CFR 1.53(b)

Filing Date Granted

An application number and filing date have been accorded to this application. The item(s) indicated below, however, are missing. Applicant is given TWO MONTHS from the date of this Notice within which to file all required items and pay any fees required below to avoid abandonment. Extensions of time may be obtained by filing a petition accompanied by the extension fee under the provisions of 37 CFR 1.136(a).

- The oath or declaration is missing.
- To avoid abandonment, a late filing fee or oath or declaration surcharge as set forth in 37 CFR 1.16(e) of \$130 for a non-small entity, must be submitted with the missing items identified in this letter.
- The balance due by applicant is \$ 130.

*A copy of this notice **MUST** be returned with the reply.*

Sandy

Customer Service Center

Initial Patent Examination Division (703) 308-1202

PART 2 - COPY TO BE RETURNED WITH RESPONSE

05/10/2001 CEH:UH 00000012 082025 09774839

01 FC:105 130.00 CH

PATENT APPLICATION

DECLARATION AND POWER OF ATTORNEY
FOR PATENT APPLICATION

ATTORNEY DOCKET NO. 10006924-1

MAY 07 2001

As a below named inventor, I hereby declare that:

My residence/post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

Recreation Of Archives At A Disaster Recovery Site

the specification of which is attached hereto unless the following box is checked:

(X) was filed on Jan 31, 2001 as US Application Serial No. or PCT International Application Number 09/774839 and was amended on _____ (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of any foreign application(s) for patent or inventor(s) certificate listed below and have also identified below any foreign application for patent or inventor(s) certificate having a filing date before that of the application on which priority is claimed:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED UNDER 35 U.S.C. 119
			YES: _____ NO: _____
			YES: _____ NO: _____

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION SERIAL NUMBER	FILING DATE

U. S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

APPLICATION SERIAL NUMBER	FILING DATE	STATUS (patented/pending/abandoned)

POWER OF ATTORNEY:

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith:

Customer Number 022879

Place Customer
Number Bar Code
Label hereSend Correspondence to:
HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400Direct Telephone Calls To:
T. Grant Ritz
(970) 898-0697

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Inventor: Shivaji GaneshCitizenship: USResidence: 230 Arriba, Apt 5, Sunnyvale, CA 940861848 Wellington Dr, Milpitas Ca 95035Post Office Address: Same as residence

Inventor's Signature

Date

4/18/2001

Rev 10/00 (DecPwr)

(Use Page Two For Additional Inventor(s) Signature(s))

Page 1 of 2

DECLARATION AND POWER OF ATTORNEY
FOR PATENT APPLICATION (continued)

RNEY DOCKET NO. 10006924-1

Full Name of # 2 joint inventor: Ramesh Kavanappillil Citizenship: IN
Residence: 19608 Pruneridge Ave Apt #9204 Cupertino, CA 95014
Post Office Address: Same as Residence
Inventor's Signature:  Date: 1/26/01

Full Name of # 3 joint inventor: _____ Citizenship: _____
Residence: _____
Post Office Address: _____
Inventor's Signature: _____ Date: _____

Full Name of # 4 joint inventor: _____ Citizenship: _____
Residence: _____
Post Office Address: _____
Inventor's Signature: _____ Date: _____

Full Name of # 5 joint inventor: _____ Citizenship: _____
Residence: _____
Post Office Address: _____
Inventor's Signature: _____ Date: _____

Full Name of # 6 joint inventor: _____ Citizenship: _____
Residence: _____
Post Office Address: _____
Inventor's Signature: _____ Date: _____

Full Name of # 7 joint inventor: _____ Citizenship: _____
Residence: _____
Post Office Address: _____
Inventor's Signature: _____ Date: _____

Full Name of # 8 joint inventor: _____ Citizenship: _____
Residence: _____
Post Office Address: _____
Inventor's Signature: _____ Date: _____

WLS



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/774,839	01/31/2001	Shivaji Ganesh	10006924-1	6266

7590 01/28/2003
HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, CO 80527-2400

EXAMINER

KINDRED, ALFORD W

ART UNIT PAPER NUMBER

2172

DATE MAILED: 01/28/2003

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 07-01)

Office Action Summary	Application No.	Applicant(s)	
	09/774,839	GANESH ET AL.	
	Examiner	Art Unit	
	Alford W. Kindred	2172	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1) ☒ Responsive to communication(s) filed on 31 January 2001.

2a) ☐ This action is FINAL. 2b) ☒ This action is non-final.

3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4) ☒ Claim(s) 1-22 is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5) ☐ Claim(s) _____ is/are allowed.

6) ☒ Claim(s) 1-22 is/are rejected.

7) ☐ Claim(s) _____ is/are objected to.

8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9) ☐ The specification is objected to by the Examiner.

10) ☐ The drawing(s) filed on _____ is/are: a) ☐ accepted or b) ☐ objected to by the Examiner.
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

11) ☐ The proposed drawing correction filed on _____ is: a) ☐ approved b) ☐ disapproved by the Examiner.
If approved, corrected drawings are required in reply to this Office action.

12) ☐ The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. §§ 119 and 120

13) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
a) ☐ All b) ☐ Some * c) ☐ None of:
1. ☐ Certified copies of the priority documents have been received.
2. ☐ Certified copies of the priority documents have been received in Application No. _____.
3. ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
* See the attached detailed Office action for a list of the certified copies not received.

14) ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application).
a) ☐ The translation of the foreign language provisional application has been received.

15) ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s). _____
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application (PTO-152)
3) ☐ Information Disclosure Statement(s) (PTO-1449) Paper No(s) _____	6) ☐ Other: _____

DETAILED ACTION

1. This action is responsive to communication: Application filed on 01/31/01.

Claim Rejections - 35 USC § 102

2. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

3. Claims 1-22 are rejected under 35 U.S.C. 102(e) as being anticipated by Tamer, US# 6,035,412.

As per claims 1-2 and 4, Tamer et al. teaches "a difference between an active data file and an archive data file to produce a delta image . . ." (see abstract) "transmitting the active data file and the delta image . . ." (see fig. 1A—sheet 1 of 15, whereas Tamer's "Bit Maps" is equivalent to applicant's "image" data) "combining the delta image with the active data file . . ." (see col. 7, lines 40-60).

As per claim 3, Tamer et al. teaches "archive data file are transaction redo logs" (see col. 2, lines 50-65).

As per claim 5, Tamer et al. teaches "storing transactions in the standby active data file . . ." (see col. 4, lines 32-49).

As per claim 6, Tamer et al. teaches "entering transactions from the archive data file into the standby transaction . . ." (see col. 2, lines 50-62).

Application/Control Number: 09/774,839
Art Unit: 2172

Page 3

As per claims 7-13, these claims are rejected on grounds corresponding to the arguments given above for rejected claims 1-6 and are similarly rejected including the following:

--Tamer et al. Teaches "mirroring device . . ." (see col. 3, lines 24-40).

As per claims 21-22, these claims are rejected on grounds corresponding to the arguments given above for rejected claims 1-2 and are similarly rejected.

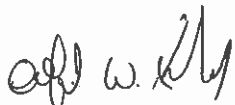
Conclusion

4. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure. US# 5,890,165.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Alford W. Kindred whose telephone number is 703-305-3802. The examiner can normally be reached on Mon-Friday, 8:00am-5:00pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kim Vu can be reached on 703-305-4393. The fax phone numbers for the organization where this application or proceeding is assigned are 703-746-7239 for regular communications and 703-746-7238 for After Final communications.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is 703-305-3900.



Alford W. Kindred
Patent Examiner
Tech Ctr. 2100
January 23, 2003

Notice of References Cited	Application/Control No. 09/774,839	Applicant(s)/Patent Under Reexamination GANESH ET AL.	
	Examiner Alford W. Kindred	Art Unit 2172	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-6,035,412	03-2000	Tamer et al.	714/6
	B	US-5,890,165	03-1999	Boudrie et al.	707/202
	C	US-			
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.



US006035412A

United States Patent [19]
Tamer et al.

[11] **Patent Number:** **6,035,412**
[45] **Date of Patent:** **Mar. 7, 2000**

[54] **RDF-BASED AND MMF-BASED BACKUPS**
[75] **Inventors:** **Phillip Tamer, Framingham; Yoav Raz, Newton, both of Mass.**
[73] **Assignee:** **EMC Corporation, Hopkinton, Mass.**
[21] **Appl. No.:** **08/820,912**
[22] **Filed:** **Mar. 19, 1997**

Related U.S. Application Data

[63] **Continuation-in-part of application No. 08/617,689, Mar. 19, 1996, Pat. No. 5,852,715.**
[51] **Int. Cl.⁷** **G06F 11/00**
[52] **U.S. Cl.** **714/6; 707/204**
[58] **Field of Search** **395/182.04; 707/204**
[56] **References Cited**

U.S. PATENT DOCUMENTS

5,161,103	11/1992	Kosaka et al. .	
5,212,784	5/1993	Sparks	395/575
5,267,148	11/1993	Kosaka et al. .	
5,276,860	1/1994	Fortier et al. .	395/575
5,276,867	1/1994	Kenley et al. .	395/600
5,495,606	2/1996	Borden et al. .	
5,555,169	9/1996	Namba et al. .	
5,561,798	10/1996	Harderle et al. .	
5,586,315	12/1996	Narang et al. .	
5,592,660	1/1997	Yokota et al. .	
5,594,900	1/1997	Cohn et al. .	

OTHER PUBLICATIONS

Koop, Peggy, "Replication at Work," DEMS, V8, n3, p54(4) (Mar. 1995).

Primary Examiner—Ayaz R. Sheikh
Assistant Examiner—Jigar Pancholi
Attorney, Agent, or Firm—Fish & Richardson P.C.

[57] **ABSTRACT**

A method of generating in a tertiary storage device a backup copy of data, the method including the steps of writing data to a first set of storage volumes; while data is being written to the first set of storage volumes, mirroring the contents of the first set of storage volumes into a second set of storage volumes; after an amount of data has been written to the first set of volumes, terminating the mirroring; while mirroring is terminated, continuing to write data to the first set of volumes thereby causing the second set of volumes to become unsynchronized from the first set of volumes; while the mirroring is terminated, backing up data stored in the second set of volumes to the tertiary storage device; after backing up the second set of volumes, resuming mirroring of the first set of volumes to the second set of volumes; and after resuming mirroring, incrementally resynchronizing the first and set sets of volumes with each other.

10 Claims, 15 Drawing Sheets

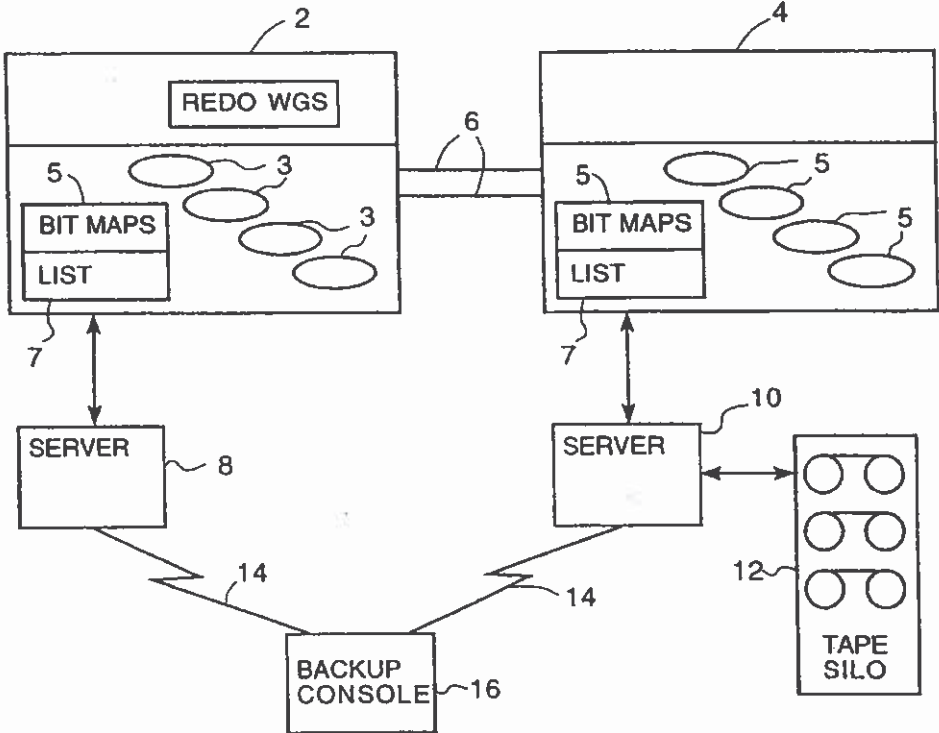


FIGURE 1A

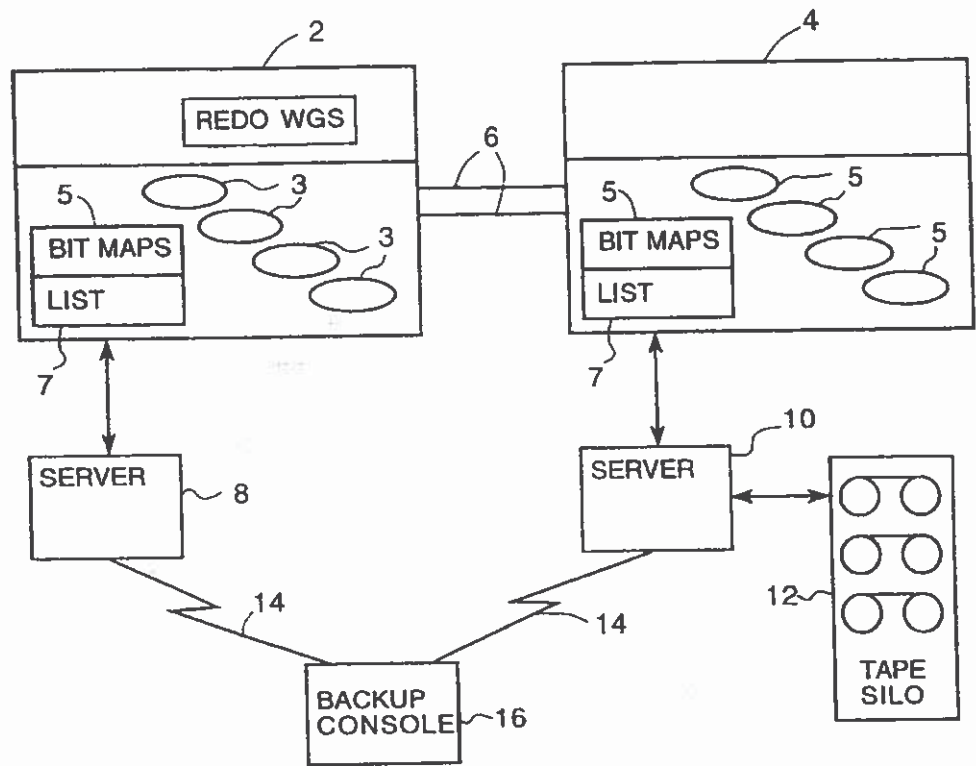


FIGURE 1B

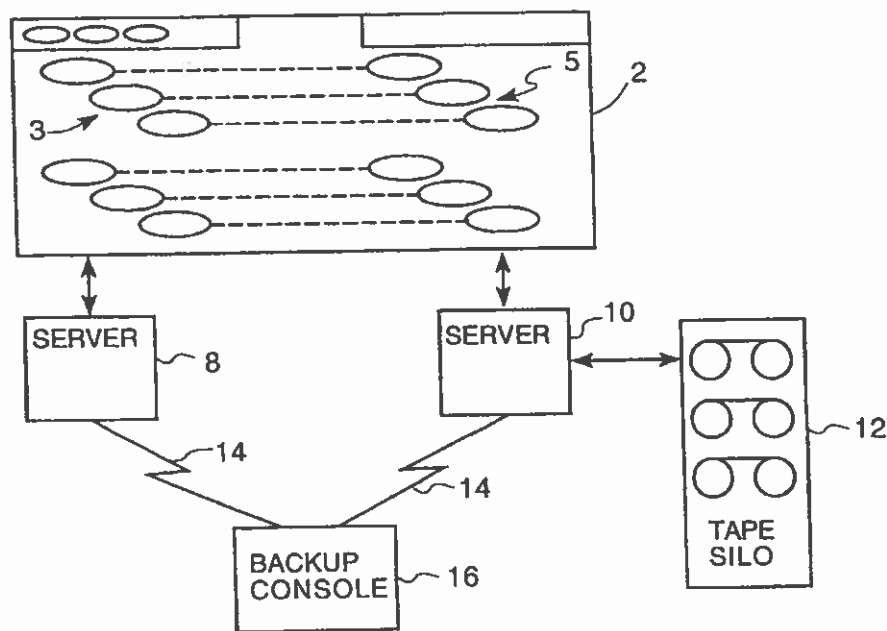


FIGURE 2

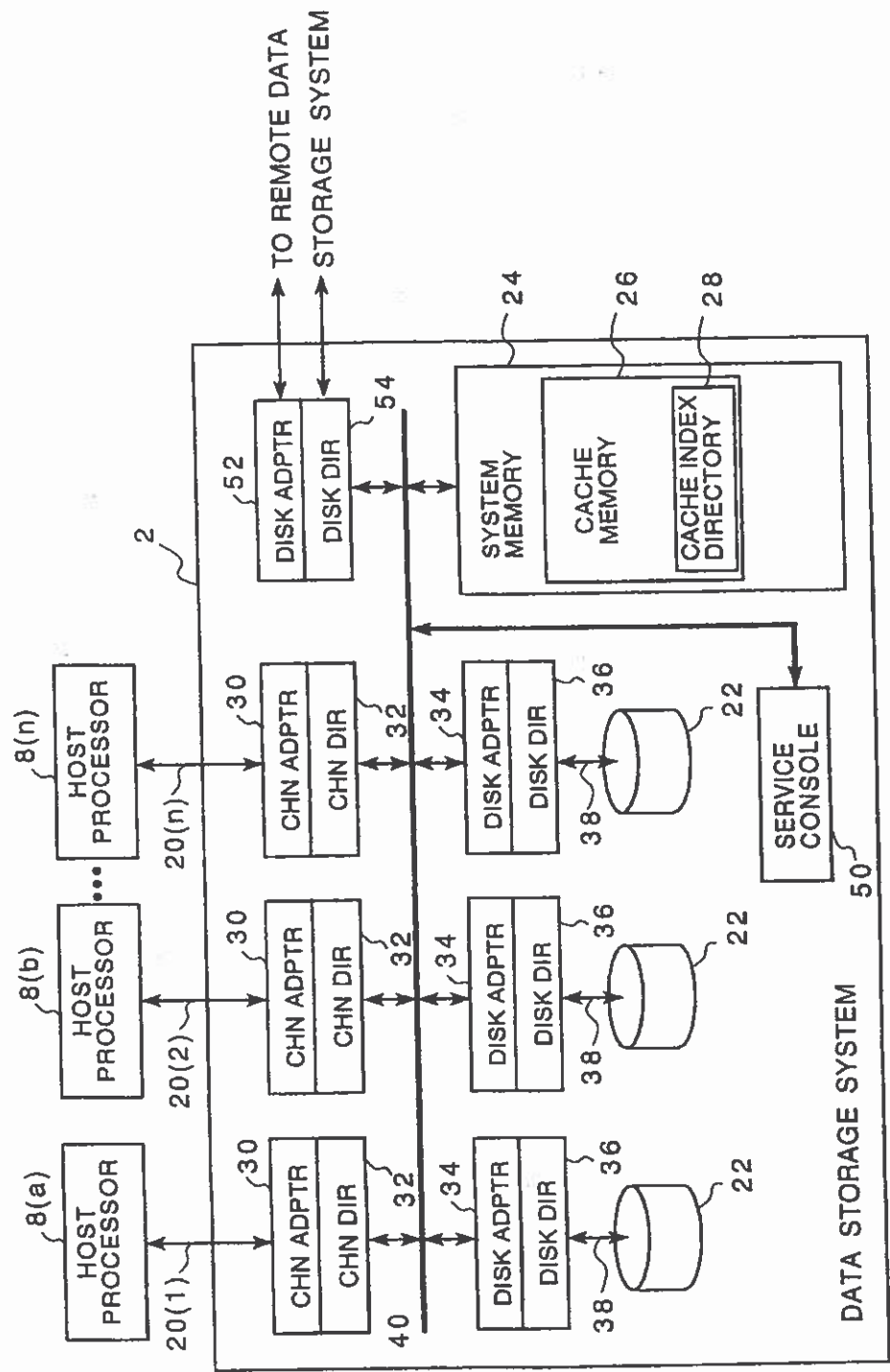


FIGURE 3



FIGURE 4A

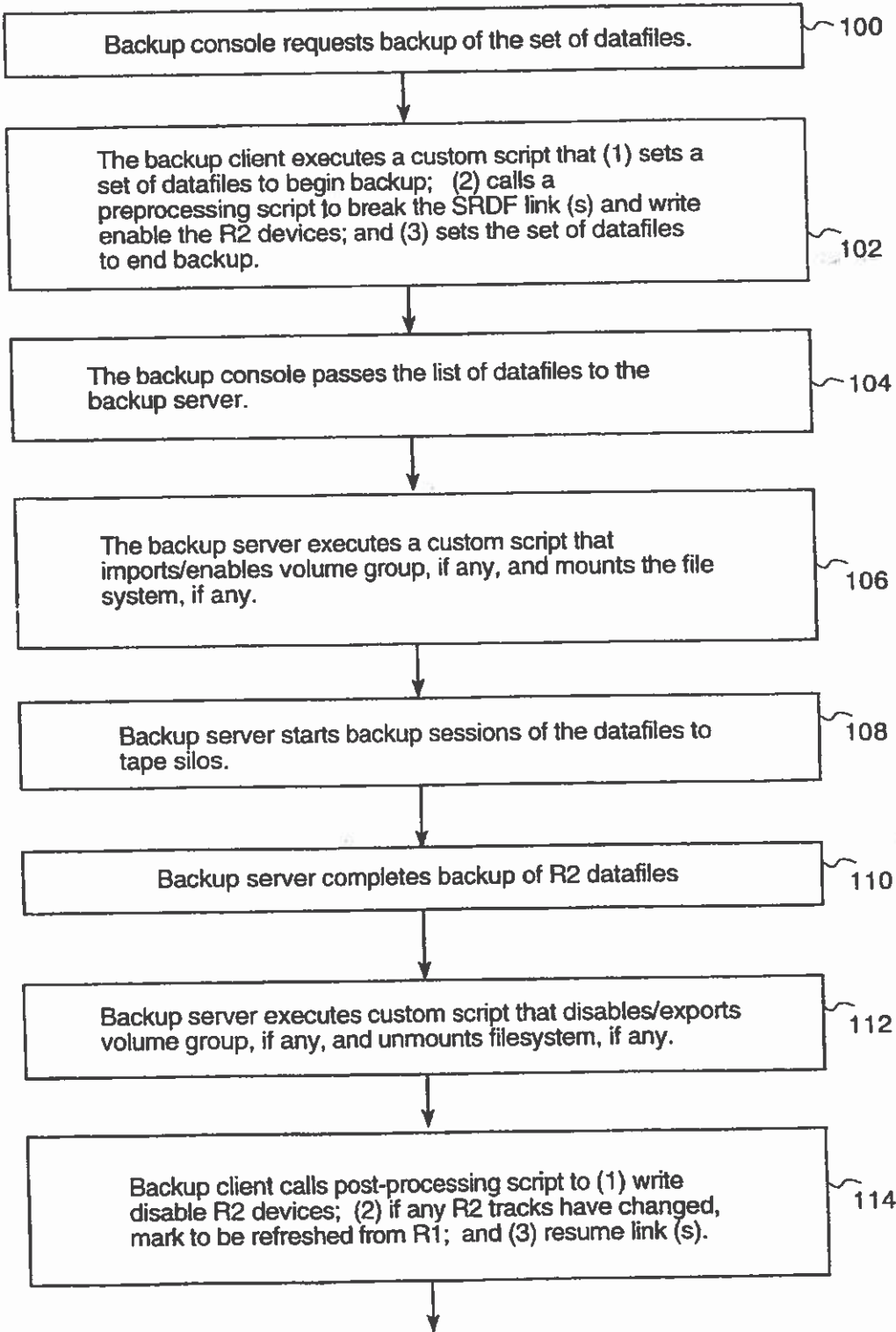


FIGURE 4B

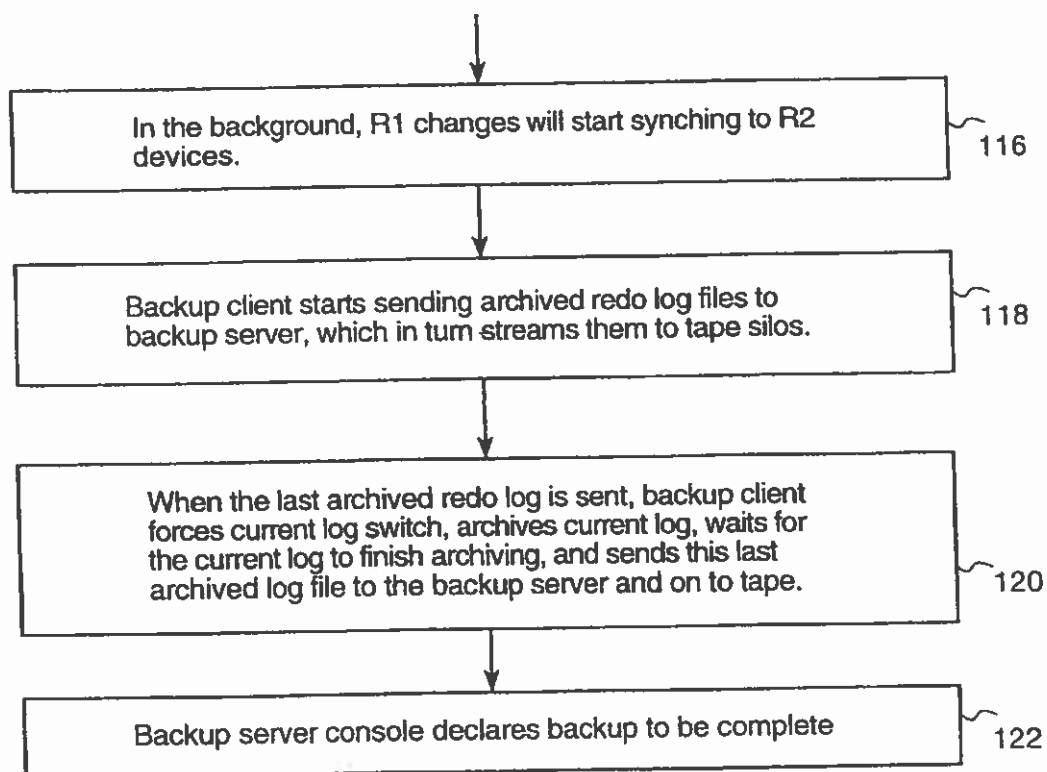


FIGURE 5

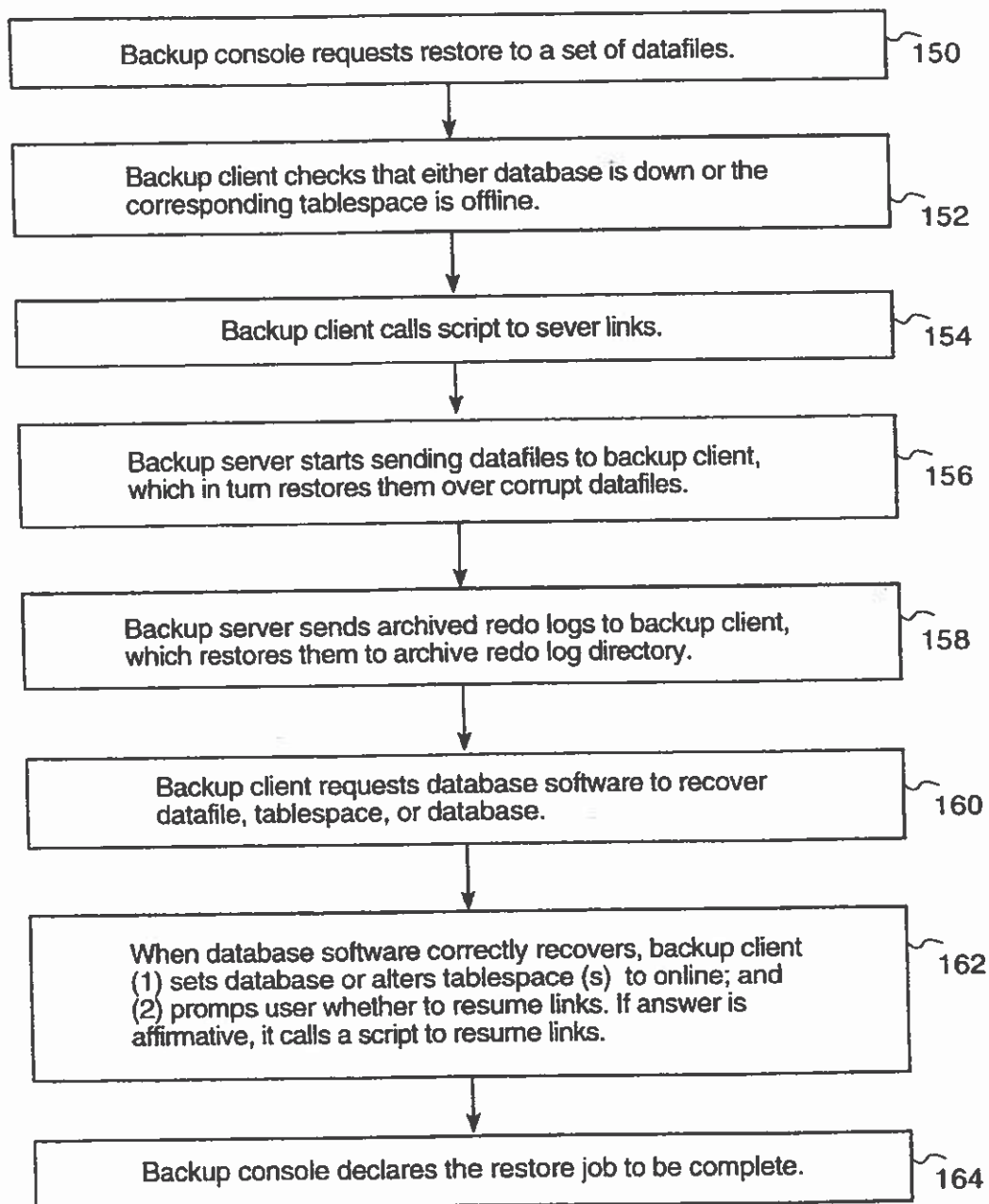


FIGURE 6A

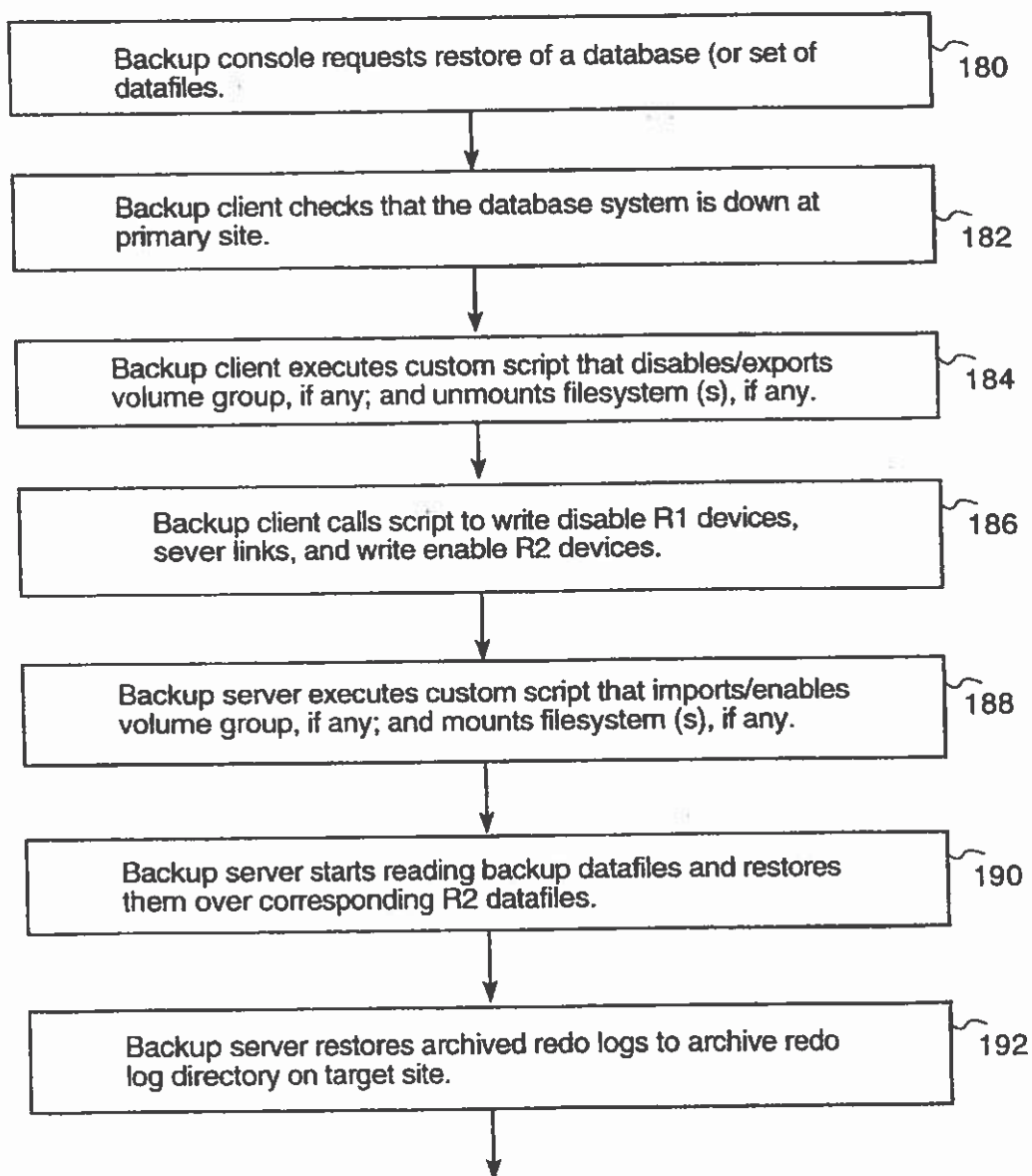


FIGURE 6B

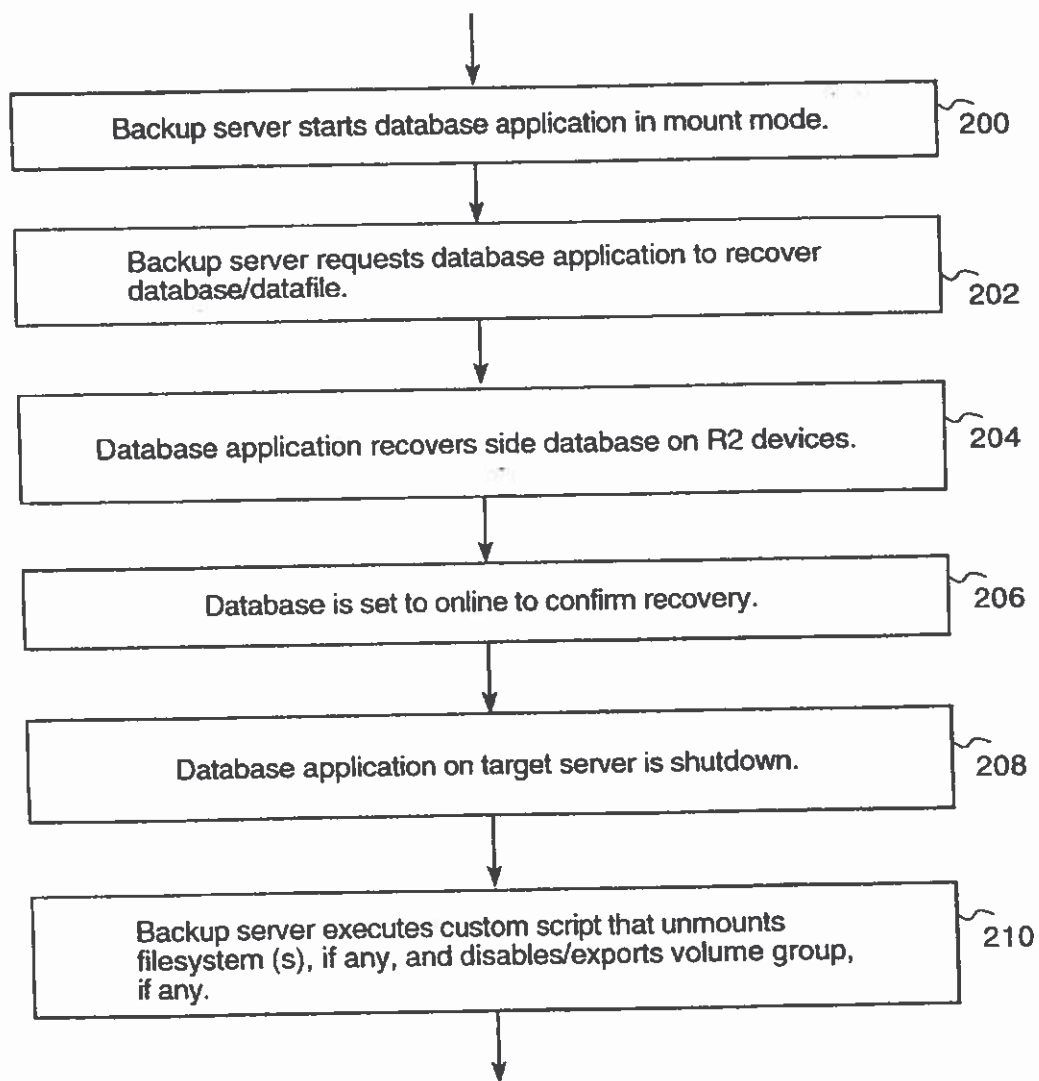


FIGURE 6C

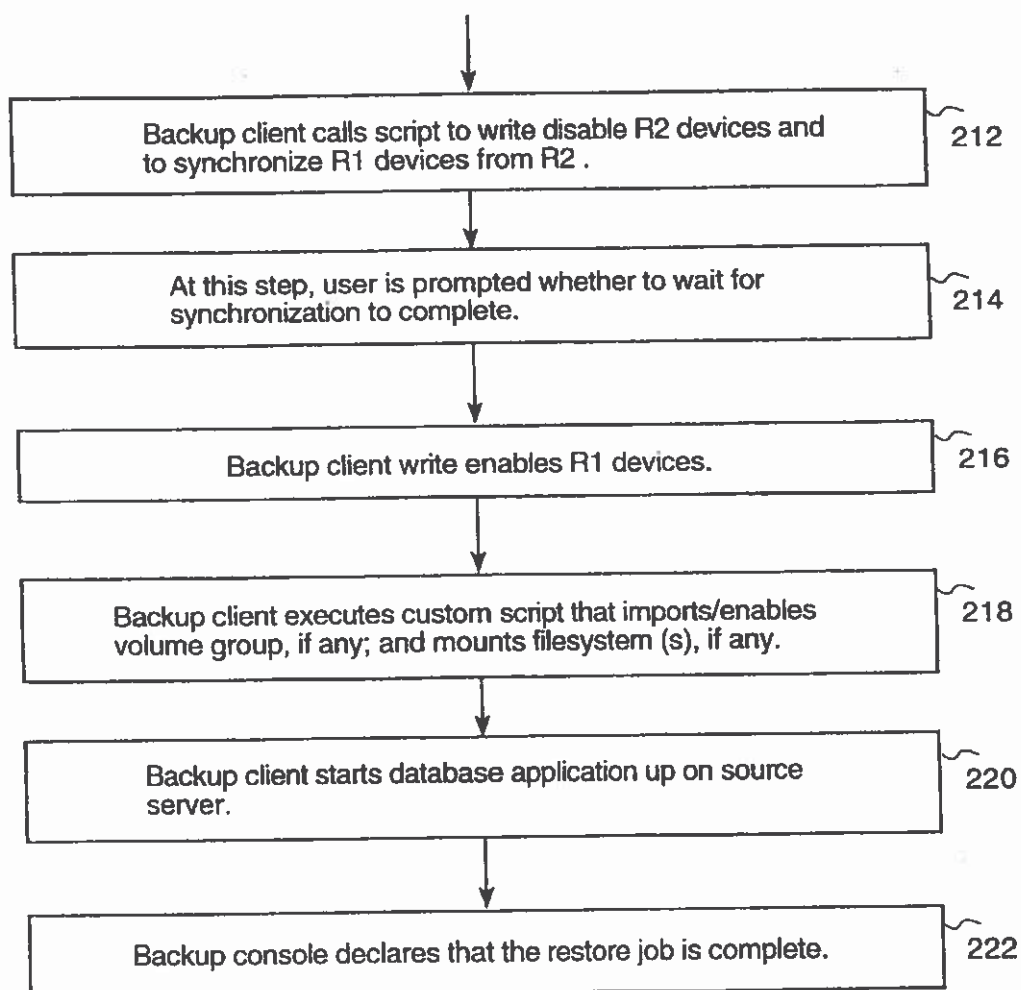


FIGURE 7A

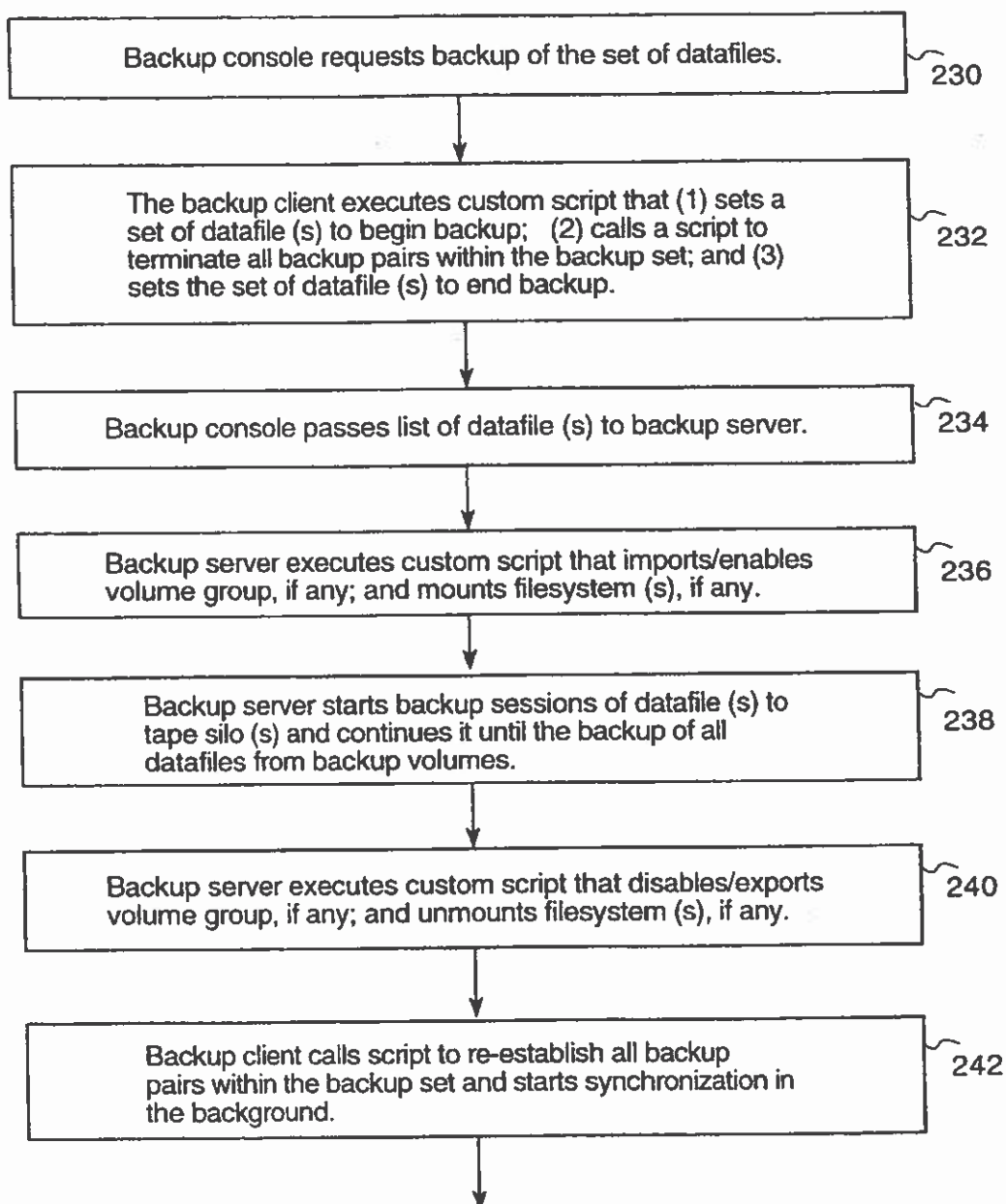


FIGURE 7B

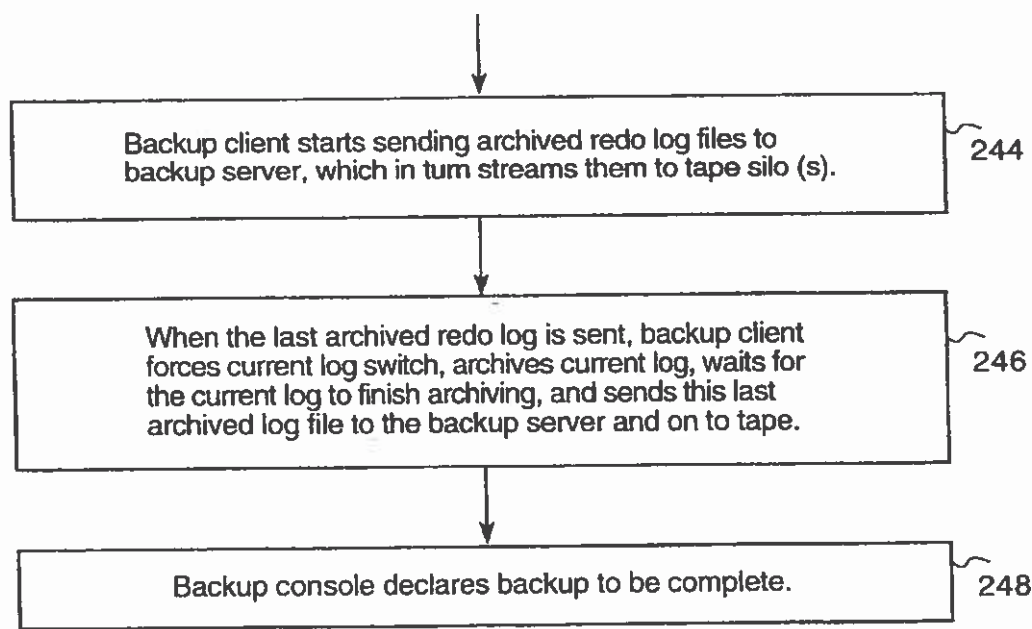


FIGURE 8

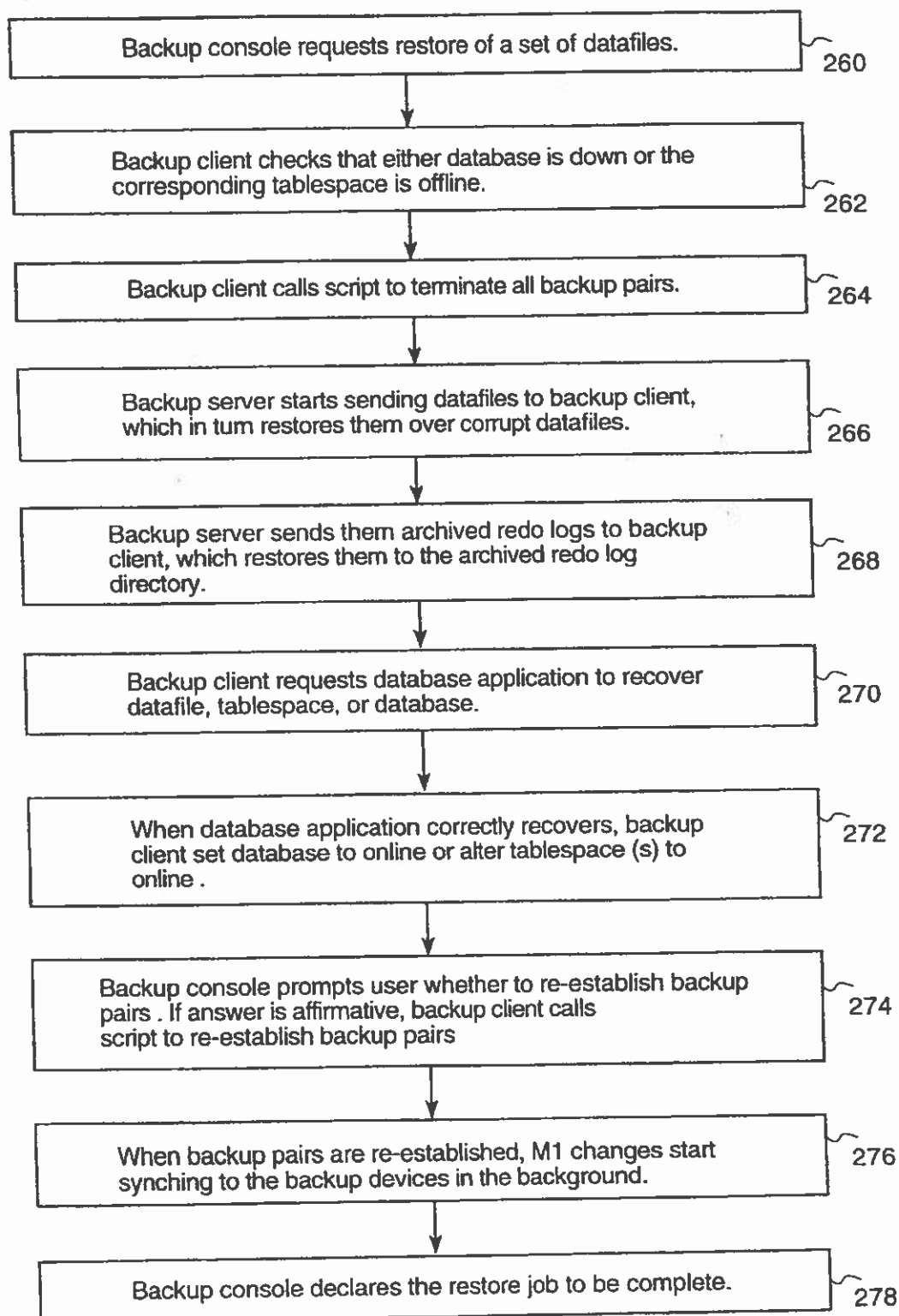


FIGURE 9A

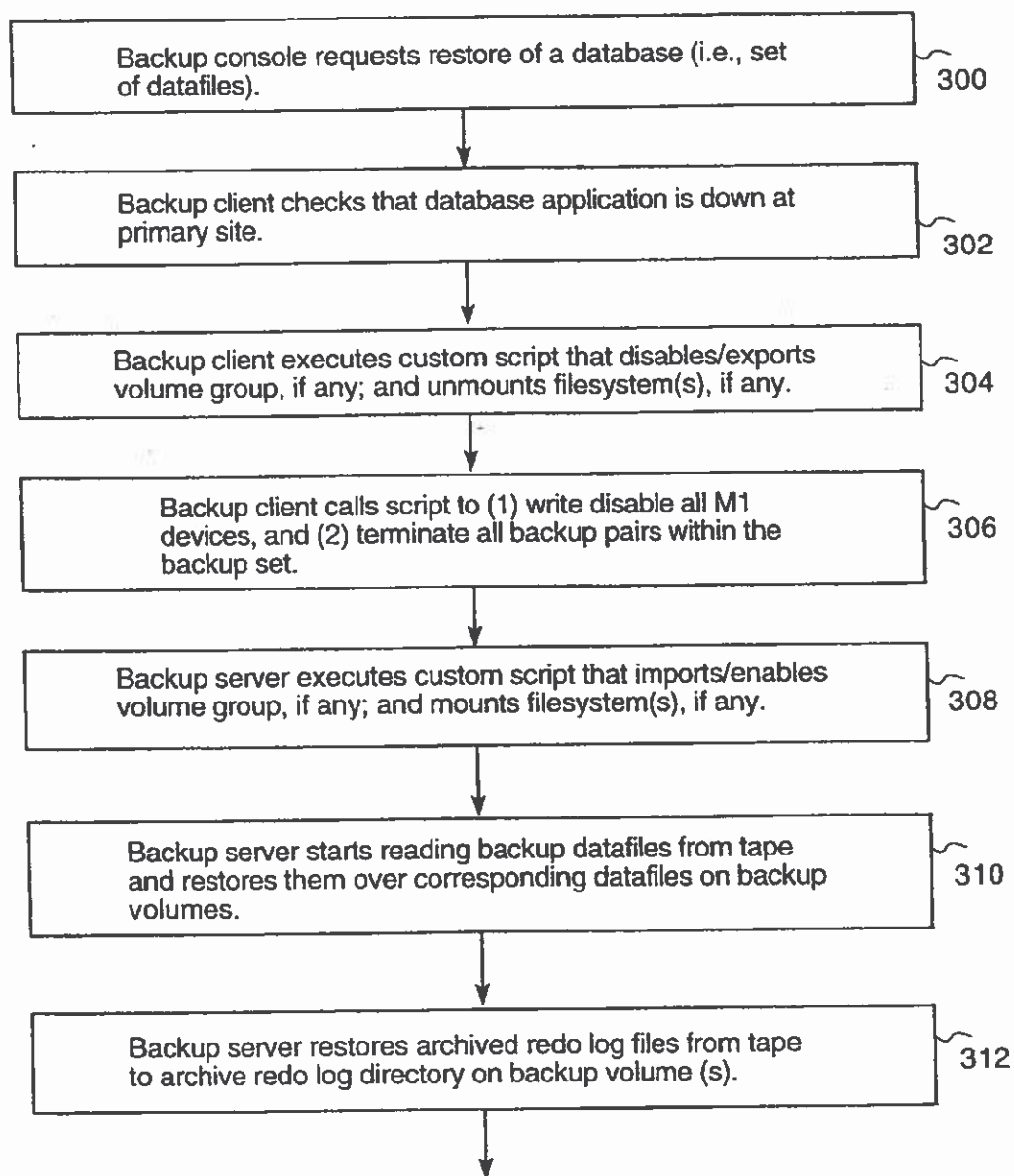


FIGURE 9B

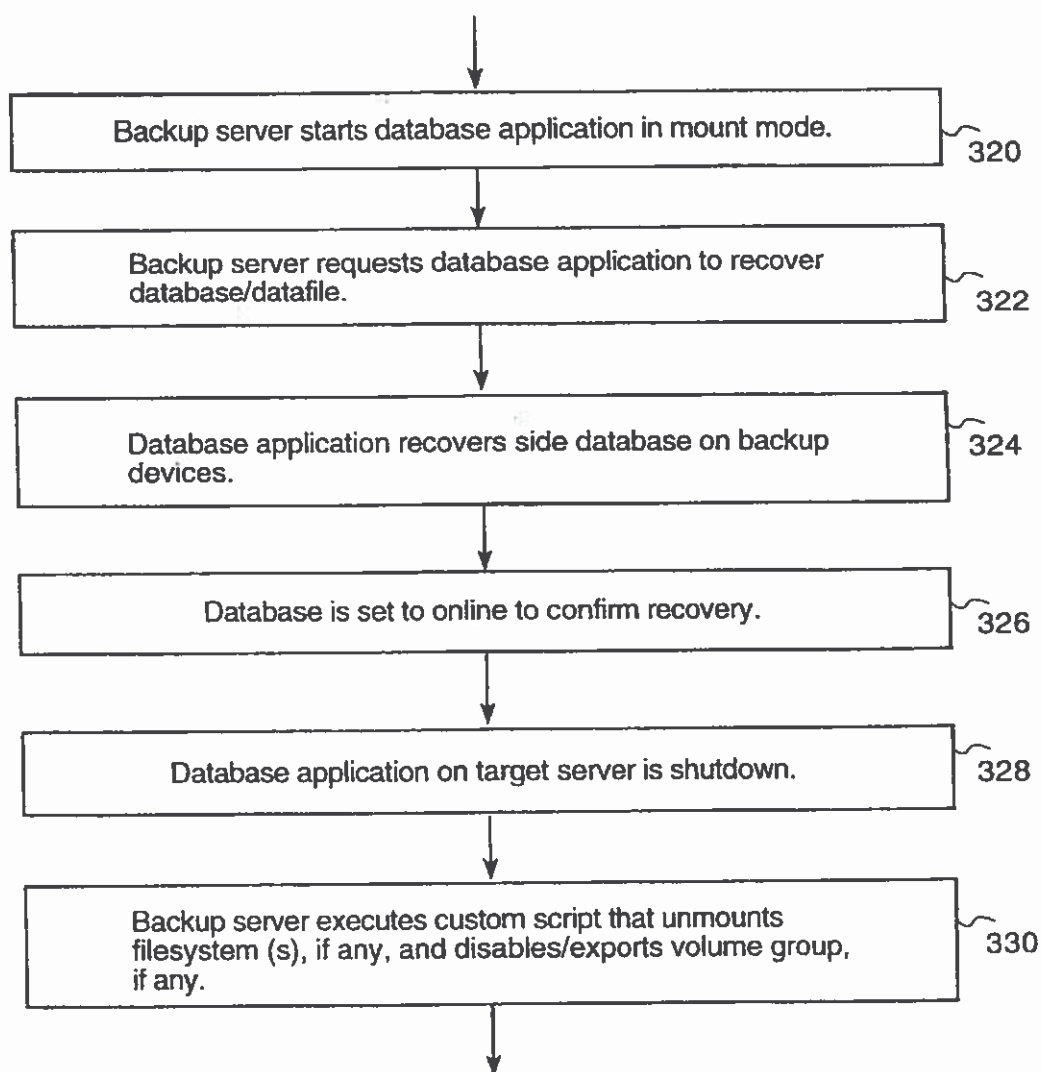
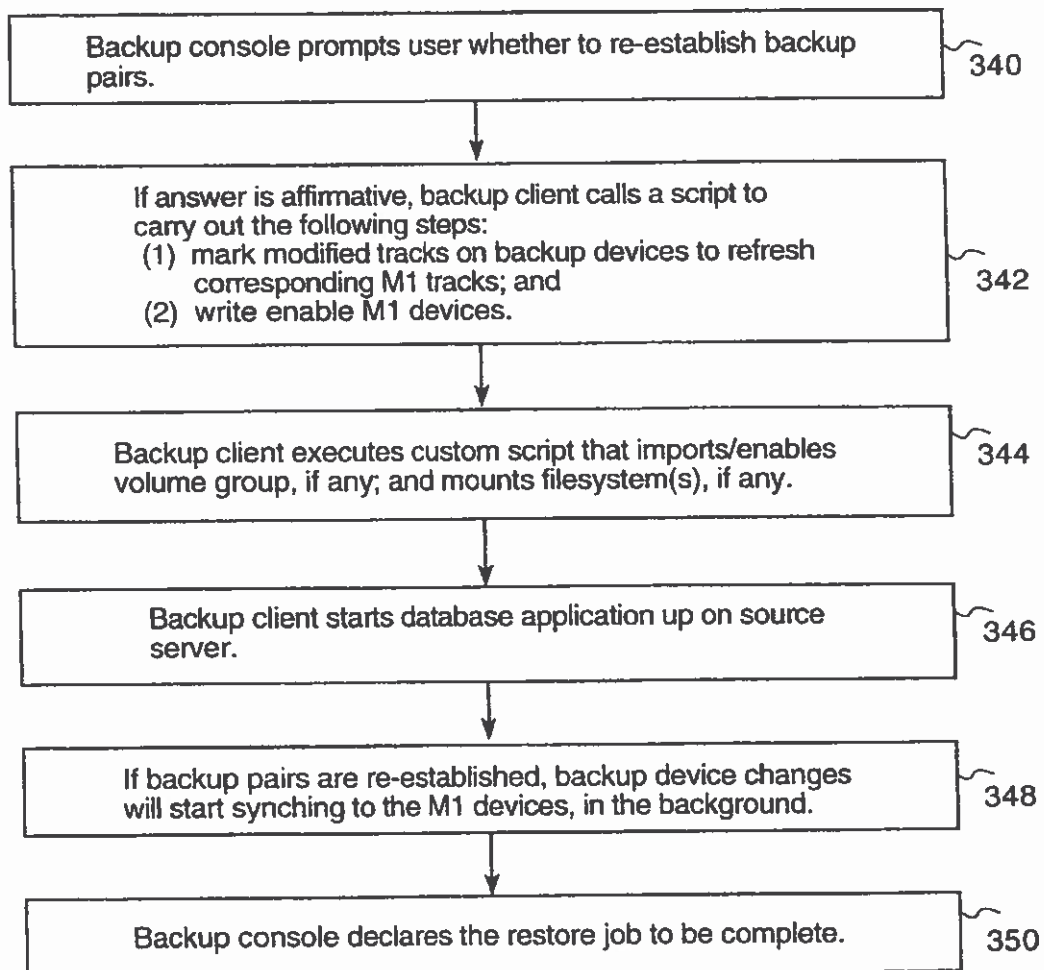


FIGURE 9C



RDF-BASED AND MMF-BASED BACKUPS

This is a continuation-in-part of U.S. patent application Ser. No. 08/617,689 now U.S. Pat. No. 5,852,715 filed Mar. 19, 1996, incorporated herein by reference.

BACKGROUND OF THE INVENTION

The invention relates generally to database backup systems and procedures and more particularly to database backup systems and procedures that are implemented using a mirror copy of the primary database.

Database systems, like many other things, are vulnerable to failures, sometimes disastrous enough to corrupt the data within the database so that it is no longer usable. To protect against such failures, database managers typically backup their data periodically. The backup operation is usually done during a window of time when loads on the database system are the lowest, e.g. late at night. If there is a window during which no work is being done, the backup can be done offline. However, if no such period exists, the backup must be done online (i.e., while other transactions are being processed). In either event, once the backup has been performed, if a failure or mistake corrupts or destroys the data on the primary online database, it can be restored from a copy that was generated during the backup operation.

Databases continue to grow in size while backup windows are shrinking. Today storage systems can handle tera bytes of data. Incremental backups for this amount of data are not usually available. Two conventional ways for doing backup are: (1) to move data through a network connected to the backup server; or (2) to run special backup software on the same machine that is running the database application. Due to bandwidth limitations of most networks, going through the network can be slow. Whereas, running backup software tends to put a considerable load on the data storage system and the host processor, e.g. the mainframe computer. While they are loaded down with handling the backup, their capability for handling business transactions goes way down. And if there is a very high volume of business transactions, the system can easily become unacceptably unresponsive. Alternatively, the backup rate goes down and the backup process will take an unacceptably long period of time. In other words, current backup solutions leave something to be desired because data is either moved through a slow network or excessive loads are placed on the database host/storage subsystem.

Two parallel avenues have been pursued for solutions. One solution path has been a database server managed backup. Data, mostly in the form of incremental changes, is passed by the database backup server to third party backup management software. The backup management software stores the data to tertiary storage. The tertiary storage can be directly attached to the database node or it can be services provided over the network. An advantage of this approach is that support for incremental backups should result in less data that needs to be backed up. Thus, loads on the resources will be reduced. Another advantage is that data integrity verification occurs at the backup times. A disadvantage is that incremental backups can reduce but they do not eliminate resource loads. Complete backups still place excessive loads on the system.

A second solution path has been to do physical backups at the datafile level. In this case, backups are performed outside of the database server. And all interactions with the database management system are done via management interfaces. The backup management software stores data to tertiary

storage with minimal or no loads on the database server/storage subsystem. That is, the paths to the tertiary storage typically bypass or minimize use of the network. An advantage of this approach is that it should result in high speed backups since loads on the database host/storage subsystem are minimal. But a disadvantage is that incremental backups are very difficult and likely not to be feasible. In addition, data integrity verification is usually not feasible at backup time.

SUMMARY OF THE INVENTION

In general, in one aspect, the invention is a method of generating in a tertiary storage device a backup copy of data. The method includes the steps of writing data to a first set of storage volumes; while data is being written to the first set of storage volumes, mirroring the contents of the first set of storage volumes into a second set of storage volumes; after an amount of data has been written to the first set of volumes, terminating the mirroring; while mirroring is terminated, continuing to write data to the first set of volumes thereby causing the second set of volumes to become unsynchronized from the first set of volumes; while the mirroring is terminated, backing up data stored in the second set of volumes to the tertiary storage device; after backing up the second set of volumes, resuming mirroring of the first set of volumes to the second set of volumes; and after resuming mirroring, incrementally resynchronizing the first and set sets of volumes with each other.

Preferred embodiments include the following features. The method further includes the steps of providing a single data storage system with a plurality of data storage volumes; and partitioning the plurality of data storage volumes into the first and second sets of storage volumes. Alternatively, the method includes the steps of providing a first data storage system; providing a second data storage system that is separate from the first data storage system; directly connecting the first and second data storage systems through a communication link; establishing the first set of storage volumes within the first data storage system; and establishing the second set of storage volumes within the second data storage system, and wherein the step of resynchronizing the first and second sets of storage volumes takes place over the communication link.

Also in preferred embodiments, the method further includes the steps of defining a first map of the first set of storage volumes; and defining a second map of the second set of storage volumes. The step of incrementally resynchronizing the first and second sets of storage volumes involves using the first and second maps to identify which portions of the second set of storage volumes need to be refreshed from the first set of storage volumes to achieve resynchronization. Also, the method includes the steps of generating archived redo logs for transactions that are applied to data stored in the first set of storage volumes; after backing up the second set of volumes, sending the archived redo logs from the first data storage system to the tertiary storage device; and storing the archived redo logs in the tertiary storage device along with the backed up data that was stored in the tertiary storage device. In addition, the method further includes providing a tape drive storage device as the tertiary storage device. Also, the method includes providing a first host processor connected to the first data storage system, wherein the first host processor is running a database application which is processing transactions and writing data from those transactions to the first set of storage volumes; providing a tape drive storage device as the tertiary storage device; and providing a second host processor connected to the second

3

data storage system, wherein tape drive storage device is connected to the second host processor which performs the backing up of data in the second set of volumes to the tape drive storage device.

Also in preferred embodiments, the first map includes a first set of submaps equal in number to the number of volumes in first set of storage volumes and each of which corresponds to a different one of the volumes in the first set of storage volumes and the second map includes a second set of submaps equal in number to the number of volumes in second set of storage volumes and each of which corresponds to a different one of the volumes in the second set of storage volumes. Each of the volumes of the first and second sets of volumes is made up of a corresponding set of tracks, and each of the submaps of the first and second sets of submaps has a number of records that is equal to the number of tracks in the set of tracks of the volume corresponding to that submap. The method also includes the steps of generating a first ordered list of identifiers of records within the first map that have changed; and generating a second ordered list of identifiers of records within the second map that have been changed using the first and second generated lists of identifiers to perform incremental resynchronization.

The invention makes it possible to do a fast online backup to tertiary storage without having to shut the system down while the backup is being performed. Mirroring is done at the physical level either within the same data storage unit or between two data storage unit that are connected to each other through a high speed communication link. Thus, when a transaction appears in the primary storage system, it also appears in the mirrored copy at basically the same time. The mirrored copy provides the copy from which the backup copy is generated.

The invention makes it possible to achieve very high backup rates with almost no effect on the primary server except for sending the archived redo logs to the backup server. And it is extensible to many other database applications. It also makes fast partial and/or complete database restores possible.

Other advantages and features will become apparent from the following description of the preferred embodiment and from the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1A is a block diagram of database system with backup capability and that utilizes one data storage unit for the primary database and a second remotely located data storage unit for a mirror copy of the contents of the physical devices that store the primary database;

FIG. 1B is a block diagram of another database system with backup capability and that utilizes a single data storage system which stores both the primary database and a local copy of the primary database;

FIG. 2 is a block diagram of the internal structure of the data storage systems that are shown in FIGS. 1A and 1B;

FIG. 3 illustrates a bit map which is used by the data storage systems to synchronize the primary database and the copy with each other;

FIGS. 4A-B present a flow chart of the steps for performing an RDF-based backup;

FIG. 5 presents a flow chart of the steps for performing a partial restore in the system shown in FIG. 1;

FIGS. 6A-C present a flow chart of the steps for performing a complete restore in the system shown in FIG. 1;

FIGS. 7A-B present a flow chart of the steps for performing an MMF-based backup;

4

FIG. 8 presents a flow chart of the steps for performing a partial restore in the system shown in FIG. 2; and

FIGS. 9A-C present a flow chart of the steps for performing a complete restore in the system shown in FIG. 1.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

The System Architectures:

Two general embodiments will be described. One embodiment, which is referred to as an RDF-based (Remote Data Facility) implementation, employs two data storage systems 2 and 4, as shown in FIG. 1A. The other embodiment, which is referred to as an MMF-based (Multiple Mirror Facility) implementation, employs a single data storage system 2, as shown in FIG. 1B.

Referring to FIG. 1A, in the RDF-based implementation, the two data storage systems 2 and 4, which may be located at geographically remote locations from each other, are connected together by a high speed data communication link 6. If they are remotely located from each other, in the event of a natural disaster or hostile act that destroys one of the data storage systems, the other data storage system will likely not be affected and can be used to support continuing database transactions.

For reasons that will become apparent shortly, data storage system 2 will also sometimes be referred to as the master unit and data storage system 4 will also sometimes be referred to as the slave unit. The master unit stores the primary database and the slave unit stores a copy from which backups to more permanent data storage are made. During a disaster recovery, the slave unit can serve as the master unit.

A server or host processor 8, which is sometimes also referred to as the backup client, is connected to data storage system 2 and is running database application, which can be a commercially available product, such as Oracle 7 by Oracle Corporation of Redwood Shores, California. The database application is continuously processing transactions and storing the results of those transactions in data storage system 2 to which its host processor is connected. Another server or host processor 10, which is sometimes also referred to as the backup server, is connected to the other data storage system 4. Its primary role in the embodiments described herein is to handle archiving of backup copies of the database to more permanent storage, such as the tape drives located in a tape silo 12 or tape farm. It also assists in the restoration and recovery of the primary database when necessary.

Both host processors are also connected to a computer network 14. In addition, there is also a backup console 16 that is connected to the same computer network and that controls and coordinates the backup, restoration, and recovery processes which are described herein.

The specific hardware that is used for the host processors and the backup console is not of great importance. Any commercially available equipment may be used. In addition, although only one host processor is shown connected to each data storage system, it should be understood that there could in fact be a plurality of host processors connected to each box.

As indicated above, the data storage systems 2 and 4 have two features that are relevant to the embodiments described herein. One feature is referred to as a remote data facility (RDF) and the other is referred to as a multiple mirror facility (MMF). Within both data storage systems 2 and 4, there is an array of storage disks 3 and 5, respectively. These disk arrays represent a set of physical volumes, typically

each volume corresponding to an individual disk. In the RDF-based implementation, the volumes which store the database files, also referred to as the source volumes (R1), are mirrored to corresponding volumes on the other data storage system, also referred to as the target volumes (R2). In other words, the data storage systems are configured so that any data that is written to any of the mirrored volumes in the master unit 2 is automatically written to corresponding volumes in the slave unit 4. The database that is stored in the R1 volumes is also sometimes referred to as M1 and the copy that is stored in R2 is sometimes also referred to as M2.

The host processor that is attached to the master unit may access the source volumes for read/write operations. However, during normal I/O operations, the target volumes cannot be accessed by the user applications that are running on the host processor that is attached to the slave unit. The transfer of the data to the slave data storage system takes place over the high speed link 6 that connects the two data storage systems and is controlled by the data storage systems.

The RDF feature also enables the user, through an appropriate set of commands, to disconnect the slave unit from the master unit by terminating the communication link and write enabling the mirror volumes in the slave unit. When that is done, the updates which occur to the master unit will, of course, no longer be sent automatically to the slave unit. However, the data that is stored in the slave unit can then be used to support other functions such as decision support, restore, and backup.

The MMF feature is similar to the RDF feature except that the mirror copy is generated within the same box as the primary database. This is done by configuring the data storage system to define two sets of devices or volumes, a first set and a second set with each of its members paired with a corresponding member in the first set. A host processor, which is connected to the data storage unit, writes its data to the first set of devices and the data storage system generates a mirror copy of that data in the second set of volumes which are paired with the first set of volumes. Thus, every write to the first set of devices also causes an automatic write of the same data to the second set of devices. In the MMF embodiment, the primary database will sometimes be referred to as M1; whereas, the mirror copy will be referred to as M2. As in the case of RDF, the host processor can instruct the data storage system to terminate the pairing between the volumes that store the primary database and the volumes that store the backup copy. When that is done, the volumes containing the backup copy are write enabled so that backup and DSS functions can be performed using the backup copy.

In the MMF-based implementation depicted in FIG. 1B, the same functionality as that described above is implemented in a single data storage system. In other words, the backup copy is generated and stored in the same box as the primary database is stored. To do this the disks are partitioned to form at least two sets, one set storing the primary database a local mirror of that database, and the other set storing the mirrored copy of the data in the volumes storing the primary database, i.e., the version from which a backup to the tape drives will be made. As in the previous embodiment, the host processors that are attached to the data storage system are prevented from writing to the backup copy while the mirror is connected internally. However, when the internal link is broken so that writes to the primary database are not automatically written to the backup copy (i.e., when the paired volumes are unpaired), then the backup copy is write enabled so that a backup to tape can be performed.

Note that in FIG. 1B there are two host processors 8 and 10 connected to the data storage system 2. However, it might also be the case that there is only a single host processor that runs both the database application as well as the backup server programs.

In the described embodiments, each of data storage systems 2 and 4 is a Symmetrix Model 5XXX or Model 3XXX Series Integrated Cached Disk Array (ICDA), though, of course, other equipment including other commercially available equipment could be used. The Symmetrix Model 5XXX and Model 3XXX ICDA's, which are products of EMC Corporation of Hopkinton, Mass., are high performance data storage systems designed for online data storage.

The following details about the internal structure and operation of data storage system 2 generally refer to the Symmetrix™ data storage systems. However, as noted above, other known architectures may also be used.

Referring to FIG. 2, data storage system 2 includes multiple arrays of disk devices 22 and a system memory 24. A portion of system memory 24 implements cache memory 26. The multiple arrays of disk devices 22 provide a permanent data storage area and cache memory 26 provides a temporary data storage area. In the currently available Symmetrix ICDA there can be up to 512 volumes, collectively capable of storing over a terabyte of data. Each disk device 22 includes a head-disk assembly, a microprocessor, and a data buffer which enables the data storage system to provide for parallel processing of data. In the described embodiment, system memory 24 is implemented by high-speed random-access semiconductor memory. Within cache memory 26 there is a cache index directory 28 which provides an indication of what data is stored in cache memory 26 and the address of that data in cache memory 26. Cache index directory 28 is organized as a hierarchy of tables for devices, cylinders, and tracks of data records, as further described in U.S. Pat. No. 5,206,939, issued Apr. 27, 1993, and incorporated herein by reference.

There is a group of channel adapters 30 and channel directors 32 that provide interfaces through which host processors 8 (e.g. host processors 8(a), 8(b), . . . 8(n)) connect to data storage system 2. Each channel adapter 30 provides for direct attachment to the physical host connections. Channel director 32, which handles I/O requests from host processors 8, contains a microprocessor that processes commands and data from host processors 8 and manages accesses to cache memory 26. It uses cache index directory 28 which is stored in cache memory 26 to determine whether the request can be satisfied out of the cache or whether the data must be obtained from disk devices 22. The particular data that is stored in cache memory is determined by the data access patterns of the data base applications that are running. Channel directors 32 write data from host processors 8 into cache memory 26 and update cache index directory 28. They also access cache index directory 26 and read data from cache memory 24 for transfer to host processors 8.

There is also a disk adapter 34 and a disk director 36 through which each disk device array 22 is connected to cache memory 26. Disk adapter 34 interfaces to multiple SCSI buses 38 to which disk device arrays 22 are connected. Disk director 36 manages accesses to the disks within disk device arrays 22. Disk Director 36 stages data from the disk device arrays to cache memory 26 and it updates cache index directory 28, accordingly. It also de-stages or writes-back data from "written-to" blocks in cache memory 26 to the disk device arrays and again updates cache index directory 28, accordingly.

Disk adapters 34 and channel adapters 30 access system memory 24 through a high-speed, parallel line system bus

40. System memory 24 is implemented by multiple memory boards. Only one accesses to any given memory board may occur at any given time, however, multiple memory boards may be accessed at the same time to support concurrent operations.

Data storage system 2 can be configured into multiple logical volumes. Typically, a volume corresponds to a single disk device. A service console 50 within data storage system 10 enables the user to configure the data storage, i.e., to define the logical volumes and to specify which logical volumes are accessible through which host connections 20.

In general, data storage system 2 can be configured to implement local mirroring, which is a feature that is in addition to the RDF and MMF features described elsewhere. When configured to implement local mirroring, the data storage system generates and maintains two copies of data by writing all modified data to two different disk devices within the same data storage system and it does this in a manner that is transparent to the host processors. Everything that is written to one volume is also written to a mirror volume and reads occur to both volumes, whichever produces the faster response. The data storage system handles write operations to a mirrored logical volume as normal write operations. That is, data is first written to cache memory. Then, the disk adapters destage the data to both devices in the mirrored pair maintaining the identical copies of the data. Further details about one particular implementation of mirroring are presented in U.S. patent application Ser. No. 07/587,253, incorporated herein by reference.

In the Symmetrix Model 5XXX Series, the high speed link 6 is implemented in accordance with the ESCON architecture of IBM, an architecture that is known to persons skilled in the art. High speed link 6 includes a dual optical fiber connection that uses serial data transmission. Independent processing units at either end of the connection control data movement from the connection to storage. More specifically, data storage system 10 includes a disk adapter 34 and a disk director 36, which couple internal bus 40 within the data storage system to high speed link 6. Similar components are present in the other data storage system.

The Bit Maps

When the backup copy of the database becomes disconnected from the primary database, which occurs during backup, for example, the two databases are likely to get out of synchronization with each other. This happens, at least in part, because the server on the master unit is permitted to continue processing transactions and updating the primary database. It will also happen because any backup to the tape drives that is performed by the backup server will also typically result in writes to the backup copy in the slave unit. Therefore, a mechanism is provided in each of the data storage systems for keeping track of where the changes occur and backing them out. According to this mechanism, the data storage system sets up and uses an array of bit maps 5 (see FIG. 3). The array includes a bit map for each of the mirrored volumes. Without this mechanism, any write that is done to the slave side would necessitate that the database be completely copied back to the slave side to establish resynchronization with the main database when the link is brought back up. Instead, resynchronization of the two databases can be done incrementally and will require only the transfer of data that has changed. The bit maps are used to identify which portions of the volumes are out of synch with their counterparts. A controller (e.g. the directors in the case of the data storage system shown in FIG. 2) in each data storage system maintains its own bit map for each volume. In the described embodiments, the highest level of

detail in each bit map is a track. In other words, referring to FIG. 3, each bit map includes a record or bucket 11 for each track t_i of the disk which it is representing and each record or bucket includes a bit for M1, corresponding to the primary database in the master unit, and a bit for M2, corresponding to the database copy in the slave unit. The controller uses the bit maps to keep track of which tracks have been modified when the backup mirroring is not operating, e.g. when the link is down in the case of the embodiment of FIG. 1A, or when the pairs within the data storage system are uncoupled in the case of the embodiment of FIG. 1B.

In the Symmetrix data storage systems, each volume can have, for example, 65K tracks, though considerably more are possible depending on how the system is configured. Thus, the bit map for each volume can be very large, including over 65K records.

In each record or bucket, a zero entry indicates that no changes have been made to the corresponding track since the link came down and a one entry generally indicates invalid data, i.e., a change has been made to the corresponding track in one database and that change was not communicated to the other database. Thus, a one in any field indicates that it is out of synchronization with the corresponding track on the other side since it contains more recent, i.e., dirty, data. Thus, each time a write occurs to a given track on the master side, the controller sets corresponding bit in the bit map (i.e., M2) to one. For example, if track N is written to, then the bit in the M2 field of the record for track N is set to one. Similarly, on the slave side, each write to the database that occurs on that side is recorded by setting the corresponding bit in the M1 field of the appropriate record in the bit map.

To optimize the resynchronization operation which occurs when the two databases, the controller also generates an ordered list 7 of modified tracks. Each entry in the ordered list 7 identifies a volume (i.e., the bit map) and a particular track within that volume. With such a list, the controller can avoid having to scan through what may be a huge number of records to find only those that have recorded a change. The immensity of that scanning task can be appreciated by looking at the numbers involved. For example, in the Symmetrix data storage systems there can be as many as 512 volumes, and each of which can have 65K tracks. The modifications which may have occurred in one disk may have only affected four tracks scattered throughout the volume. To find the identity of the tracks that have been modified through scanning requires checking all of those 65K records. With the list of modified tracks, however, the controller can jump directly to the relevant entries in the relevant bit maps.

As will become apparent in the following description, the synchronizing which takes place between R1 and R2 (or M1 and M2) after a link has been brought back up takes place in the background. While it takes place, with the aid of the bit maps, both systems will know where the best data is. Thus, the database activity can be directed to valid tracks whether they are in R1 or R2.

Another use of the bit maps arises when a physical disk fails. Suppose the system does not have any local disk protection (e.g. local mirroring) and a disk fails. In that event, the system can simply set all of the M2 bits for the track that has failed and redirect activity involving that disk to the backup copy in the slave unit (or the paired volumes). When the disk is replaced, the bit map is then used to synchronize the new disk with its corresponding disk in the slave unit (or with the paired disk on the same box).

Overview of Backup and Restore Operations:

In general, the system is set up to mirror a set of one or more volumes both internally and to the remote data storage system. The mirroring to the remote system occurs over the high speed link. When it is time to prepare a backup, the link is cut by software and the mirroring to the remote data storage system is halted.

Prior to cutting the link, it should be noted that the database in the master unit is R/W enabled but the database copy in the slave unit is not W enabled. Thus, while the link is up, no host processor connected to the slave unit can write to the volumes that contained the copy of the data in the corresponding volumes on the master unit. However, after the link is brought down, the volumes on the slave side are W enabled to permit access the table space during the backup operation. When a table access occurs, typically a write also takes place. These writes are all recorded in a bit map that is maintained on the slave unit side.

Note, that until the backup to tertiary storage is completed, writes to the data in the database copy stored in the slave unit are not permitted. Any writes to data prior to completing the backup would render that data invalid. After the backup to tape storage is complete, however, then writes to the data may take place, e.g. as in the case when the disconnected copy is used to support decision support functions.

The backup client can still continue to make changes to the primary database but those changes will no longer be communicated to the remote data storage system, at least for the period during which the link remains down. However, the local host processor keeps track of all changes that occur to the primary database by logging all of them in an archived redo log file.

The backup is performed using the mirrored copy in the remote data storage system. When the backup operation at the remote data storage system is finished, the backup client moves all of the archived redo logs from the database server to the backup server. Since this transfer takes place over the network, it will generate a load on the network, but the load is minor in comparison to the alternative of sending all backup data over the network at the time of backup.

When the archived redo logs are completely transferred to the backup server and moved into the backup facility, e.g. the tape silo, the backup is finished. At that point the link is reestablished and the main disks and the mirrored disks are incrementally resynchronized, i.e., only the changes that were made to the databases while the link was down are transferred. The incremental resynchronizing of the two groups of disks (i.e., the two databases) takes place in the background and transparently to the host processors. The resynchronization is done by sending only the changes which occurred in the main system to the backup system along with any tracks that have to be refreshed in the slave due to writes which took place there during the backup procedure. To aid in this process, a log is maintained of the changes that occurred to the primary database (M1) and the backup database (M2) when the link was down. In essence any changes that were done to the primary database (M1) are sent over to the mirrored side and any changes that occurred to the backup database (M2) when the link was down are erased (by overwriting them with data from corresponding portions of the primary database). Thus, instead of having to rewrite the entire database to the slave side to resynchronize the two databases with each other, the incremental resynchronization technique enables one to only transfer the data for tracks that were modified while the link was down. This greatly decreases the amount of data that needs to be transferred.

With decision support, it is necessary to make the database consistent, but this is not required with the backup copy. Therefore, the redo logs are simply saved and used later to achieve consistency during the restore operation, as described in greater detail below.

Typically a file system check will be done by the backup server when it is saving the data to tertiary storage. This involves writing to the same volumes that store the database (but not to the stored data itself), which, of course, produces dirty blocks or tracks. In the embodiments described, the data storage systems keep track of which blocks or tracks have been changed so that the changes can be erased or undone during the restore operation.

So when the link is reestablished, the two sides are resynchronized. This involves first synchronizing the bit maps so that both sides have the same views. This is done by flipping the bit map on the slave side (i.e., by transferring M1 information to M2 and by transferring M2 information to M1) and then combining the results with the bit map on the master side through a simple OR'ing operation. The result is a bit map on both sides that identifies all tracks that must be sent over to the slave side to achieve resynchronization. Then, the data storage system containing the primary database begins sending the marked tracks to the other side, i.e., all tracks which are identified as having been modified on the master side and all tracks which were identified as having been modified on the slave side. As each track on the other side is refreshed or updated, the corresponding bits are cleared.

RDF-Implemented Backup

FIGS. 4A-B illustrate in greater detail the steps that are performed in creating a backup in the embodiment shown in FIG. 1A.

The backup console begins the process by requesting a backup of a specified set of datafiles (step 100). It is, of course, assumed that all of the database datafiles, except for the archived redo log directory, are on volumes that are being mirrored to the slave data storage system over the high speed link. Actually, if this setup is only for backup purposes only, i.e., no disaster tolerance or decision support are required, then the online redo log files do not have to be on the mirrored volumes either.

In response to receiving the backup request, the backup client executes a custom script that performs a sequence of operations (step 102). First, it sets a set of datafiles to begin backup. Then, it calls a pre-processing script within the data storage system to shut down the high speed link and write enable the R2 devices. Finally, it sets the set of datafiles to end backup.

The backup console passes the list of datafiles to the backup server so that the backup server can begin to perform the actual backup operation (step 104). In response, the backup server executes a custom script that imports or enables the volume group(s), if any, and it mounts the file system(s), if any (step 106). Once that is completed, the backup server starts the backup session during which it sends the identified R2 datafiles to the tape silos (step 108).

During this period when the link is down, both data storage systems use their bit maps to identify of all tracks within their volumes that are modified. For example, for each track that is modified on the master side, the master unit enters a one in the M2 field for the track of the appropriate bit map (i.e., the bit map for the volume that was modified). Similarly, on the slave side, all writes to the mirror disks are identified by entering ones in the M1 fields for the tracks of the appropriate bit maps. The bit entries on the master side identify which tracks need to be sent over to the slave side

to bring the slave side up to date with the master side. And the bit map entries on the slave side identify which tracks need to be refreshed from the R1 to overwrite the changes that occurred during backup to tape.

When the backup server completes the backup of the R2 datafiles (step 110), it executes custom script that disables or exports the volume group(s), if any, and unmounts the file system(s), if any (step 112).

The backup client then calls a post-processing script (API) which when executed instructs the data storage systems to which carry out a number of functions (step 114). It write disables the R2 devices. It also determines whether any R2 tracks have changed, and if they have it marks them to be refreshed from R1. And it re-establishes the high speed link(s) connecting the slave data storage system to the master data storage system.

Once the link has been reestablished, the R1 changes start-synching to R2 devices (step 116). This resynching operation is carried out in the background and uses the bit maps to determine which tracks in R2 need to be updated by data from corresponding tracks in R1. It begins with first synchronizing the bit maps so that they are the same on both sides of the connection. That is, both units will have the same view identifying on which side the valid data resides. The synchronization of the bits maps is accomplished by flipping the bit maps on the slave side and then combining those bits maps with the corresponding bit maps on the master side. By flipping, we mean that the entries in the M1 side of the bit map are transferred to the corresponding track on the M2 side of the bit map. The result identifies all tracks in R1 that need to be sent to R2 to accomplish one of two things: (1) to forward updates from R1 to R2 which occurred as a result of the continuing transaction processing on the master while the link was down; and (2) to overwrite any modifications that might have been made to R2 while the link was down and as a result of the backup operation.

As the marked tracks are sent to R2 from R1, the data storage units clear the corresponding entries in their respective bit maps.

The backup client also starts sending its archived redo log files over the network to backup server (step 118). And the backup server, in turn, streams them to the tape silos.

When the last archived redo log is sent, the backup client forces a current log switch and it archives the current log. As soon as the backup client finishes archiving the current log, it client sends this last archived log file to the backup server which passes the log on to tape (step 120).

At that point, the backup console declares backup to be complete (step 122). No new backup session is permitted to start until the R1 and R2 devices have been fully synched via the high speed link.

RDF-Based Restores

After a backup has been completed in the manner described above, there may arise a need to perform either a partial or a complete restore of the database on the master data storage system. Each of the processes used to accomplish this will now be described.

A Partial Restore

Referring to FIG. 5, to start a partial restore of the database in the master unit, the backup console requests restore to an identified set of datafiles (step 150). In response, the backup client confirms that either the database is down or the corresponding tablespace is offline (step 152). After confirming that either of these conditions exist, the backup client calls script which when executed instructs the data storage system to bring down the high speed link that connects it to the slave unit (step 154).

Once the high speed has been terminated, the backup server starts sending the identified datafiles to the backup client over the computer network (step 156). The backup client, in turn, restores them over the corrupted datafiles on its data storage unit. Once the identified datafiles have been transferred to the backup client, the backup server then sends the archived redo logs and the backup client restores them to archive redo log directory (step 158).

After restoring the archived redo logs to the directory, the backup client requests the database software to recover the datafile, tablespace, or database, whichever is appropriate (step 160). All of the changes that are made to R1 are recorded in the bit maps by entering 1's in the appropriate tracks for the R1 side. Thus, when the recovery is complete, the bit maps on the master side will identify all tracks of R1 that need to be sent to R2 to synchronize R2 with R1.

After the database software correctly performs the recovery, the backup client sets database to online or alters tablespace(s) to online; and it then prompts the user whether to reestablish the high speed link (step 162). If the user answers in the affirmative, the back client calls a script which when executed instructs data storage unit to bring up the high speed link (step 162). Once that is accomplished, the backup console declares the restore job to be complete (step 164).

With the link re-established, the master unit synchronizes R2 with R1. It does this by transferring all of the restored tracks, as identified in the bit maps, to R2.

A Complete Restore

In the event that both databases have been corrupted (e.g. through an unintended deletion of records), it may be necessary to start a complete database restore on the slave side while the primary database is offline. In that case, the backup console requests a restore of the database (e.g., a set of datafiles) (step 180) (see FIGS. 6A-B). In response, the backup client checks that the database software is down at primary site (step 182). And, assuming that it is, the backup client then executes custom script that disables/exports the volume group(s), if any, and unmounts the file system(s), if any (step 184). The backup client then calls a script which when executed instructs the data storage systems to write disable the R1 devices, to bring down the high speed link, and to write enable the R2 devices (step 186).

The backup server then executes a custom script that imports/enables the volume group(s), if any, and mounts file system(s), if any (step 188). Then, the backup server starts reading the backup datafiles from the tape drives and restores them over the corresponding R2 datafiles in the slave unit (step 190). After the datafiles have been written to the slave unit, the backup server restores the archived redo logs to archive redo log directory on the target site (step 192).

With the backup copies of the datafiles and the archived redo logs now available in the data storage system, the process for recovering the side database on target site begins. First, the backup server starts the database application software in mount mode (step 200). Then, the backup server requests the database software to recover the database/datafile (step 202). And in response, the database software recovers the side database on the R2 devices (step 204). With the recovery complete, the database application is set to online to confirm recovery (step 206). Once confirmed, the database application on the target server is shutdown (step 208). And the backup server then executes custom script that unmounts the file system(s), if any, and disables/exports the volume group(s), if any (step 210).

It should be understood that the recovery could also take place on the source side by simply sending the backup copies and the archived redo logs over to the of the source side.

With the database restored in the secondary data storage system, the offline database on the master side is then restored. This is done by the backup client calling a script which when executed instructs the data storage systems to write disable the R2 devices, bring up the high speed connection between the master and the slave unit, and then synchronizes the R1 devices from the R2 devices (step 212).

The bit maps are used to perform the synchronization and to determine where the valid data is throughout the process. Prior to establishing the link, the slave unit enters ones in the M1 side of its bit map to indicate that all tracks on the slave side have the most recent or valid data. When the link is reestablished, the bit maps on the master side are synchronized with those on the slave side so that they both have the same view of the data. Then during synchronization of R1 with R2, all tracks that are marked in R2 (i.e., the tracks for which the M1 field contains a one) are transferred to R1 and the appropriate locations in the bits maps are cleared to indicate that the tracks are synchronized.

At this point, the user is prompted whether to wait for synchronization to complete (step 214). The backup client then write enables the R1 devices (step 216). It also executes a custom script that imports/enables the volume group(s), if any, and mounts the file system(s), if any (step 218). Next, the backup client starts the database application on the source server (step 220). At that point, the backup console declares that the restore job is complete (step 222).

Note that the backup client need not wait for synchronization to complete before commencing with processing of further transactions. Since the bit maps tell the master unit where the valid data is, the master unit can direct the transactions to the appropriate one of the two data storage systems. For example, if a transaction requires a read of or a write to a particular track that has not yet synchronized with R2 (as indicated by a 1 in the M2 side of the bit map), the master unit can direct the read/write to the corresponding track in R2.

MMF-Implemented Backup:

FIGS. 7A-B illustrate in greater detail the steps that are performed in creating a backup in the embodiment shown in FIG. 1B.

In this embodiment, every volume containing the database, except the archived redo log directory, is paired with a backup volume within the same data storage unit. The backup volumes constitute a backup set and they are not available to any host processor at this time. Actually, since the setup which is to be described is only for backup (i.e., no decision support is required), the online redo log files should not be on the paired volumes.

The backup console begins by requesting a backup of the set of datafiles (step 230). In response, the backup client executes a custom script that (1) sets a set of datafiles(s) to begin backup; (2) calls a script within the data storage system to terminate all backup pairs within the backup set; and (3) sets the set of datafile(s) to end backup (step 232). At this point, the backup volumes are available to the backup server/host.

The backup console then passes a list of datafile(s) to backup server (step 234).

Next, the backup server executes a custom script that imports/enables volume group(s), if any, and mounts the file system(s), if any (step 236). After this, the backup server starts backup sessions of the datafile(s) to the tape silo(s) until all datafiles from the backup volumes have been transferred (step 238). The backup server then executes another custom script that disables/exports the volume group(s), if any, and unmounts the file system(s), if any (step 240).

The final phase of the online backup begins with the backup client calling a script which when executed instructs the data storage system to re-establish all backup pairs within the backup set (step 242). This again makes the backup volumes not available to the backup server/host. In the background, any changes that were made to the M1 devices start synching to the backup devices M2.

As in the case of the RDF-implemented backup described above, bit maps are also used to perform the synchronization. There is a set of bit maps for M1 and another set for M2, just as there was a set of bit maps for R1 and another set of bit maps for R2. And the data storage system uses the bit maps in precisely the same way as was described previously.

The backup client starts sending archived redo log files to the backup server, which in turn streams them to tape silo(s) (step 244). When the last archived redo log is sent, the backup client forces a current log switch, archives the current log, waits for the current log to finish archiving, and then sends this last archived log file to the backup server, which in turn sends it on to tape (step 246).

Alternatively, the backup server could be given direct access to the archived redo logs stored in the data storage system, and it could send them to the tape storage without the need of the backup client being involved.

With the current redo log on tape, the backup console declares backup to be complete (step 248). As in the previous example, no new backup session can start until M1 and the backup devices (i.e., M2) are fully synched.

MMF-Based Restores

After a backup has been completed in the manner described above, either a partial or a complete restore can be performed. Each of these operations will now be described.

A Partial Restore

Referring to FIG. 8, to begin a partial restore to an online database, the backup console requests a restore of a set of Datafiles (step 260). In response, the backup client checks that either database is down or the corresponding tablespace is offline (step 262). Then, the backup client calls a script which when executed instructs the data storage system to terminate all backup pairs (step 264).

Once the pairs have been terminated, the backup server starts sending datafiles over the network to the backup client. And the backup client, in turn, restores them over the corrupted datafiles (step 266). After the corrupted files have been replaced, the backup server sends the archived redo logs to backup client, which in turn restores them to the archived redo log directory (step 268). Alternatively, the backup server could be given access to the archived redo log directory, in which case, it could send the archived redo logs to that directory without the assistance of the backup client.

The backup client then requests the database application to recover the datafile, tablespace, or database (step 270). After the database application completes the recovery, the backup client sets the database or alters the tablespace(s) to online (step 272).

With this completed, the backup console prompts user whether to re-establish the backup pairs (step 274). If the answer is affirmative, the backup client calls a script which when executed instructs the data storage system to reestablish the backup pairs (step 274). If backup pairs are re-established, M1 changes start synching to the backup devices in the background (step 276). Again, at this point, the backup volumes will no longer be available to the backup server/host. And the backup console declares the restore job to be complete (step 278).

A Complete Restore

The method for performing a complete restore to a side database while primary database is offline is depicted in FIGS. 9A-C.

It begins by the backup console requesting a restore of a database (e.g. a set of datafiles) (step 300). In response, the backup client checks that database application is down at primary site (step 302). Assuming that it is, the backup client then executes a custom script that disables/exports the volume group(s), if any, and unmounts file the system(s), if any (step 304). And the backup client calls a script which when executed instructs the data storage system to write disable all M1 devices, and to terminate all backup pairs within the backup set (step 306). At that point, the backup volumes become available to the backup server/host.

The backup server executes a custom script that imports/enables the volume group(s), if any, and mounts the file system(s), if any (step 308). Next, the backup server starts reading backup datafiles from tape and restores them over corresponding datafiles on the backup volumes (step 310). When the reading of backup datafiles is complete, the backup server restores the archived redo log files from tape to the archive redo log directory on the backup volume(s) (step 312).

This establishes the conditions required for recovering the side database on the backup volumes. So, referring to FIG. 9B, the backup server starts the database application in mount mode (step 320). And it requests the database application to recover the database/datafile (step 322). In response, the database application recovers the side database on the backup devices (step 324). After finishing recovery, the database is set to online to confirm recovery (step 326). Once recovery is confirmed, the database application on the target server is shutdown (step 328). And the backup server executes a custom script that unmounts the file system(s) if any, and disables/exports the volume group(s), if any (step 330).

Referring to FIG. 9C, the process of restoring to an offline database is finished by the backup console prompting the user whether to re-establish backup pairs (step 340). If answer is affirmative, the backup client calls a script which when executed instructs the data storage system to perform a number of functions (step 342). First, it marks modified tracks on the backup devices to refresh the corresponding M1 tracks. Then, it re-establishes the backup pairs and write enables the M1 devices. The backup client executes a custom script that imports/enables the volume group(s), if any, and mounts the file system(s), if any (step 344). Finally, the backup client starts up the database application on source server (step 346).

If backup pairs are re-established, the backup device changes start synching to the M1 devices, in the background (step 348). Thus, the backup volumes are again not available to the backup server/host. And the backup console declares the restore job to be complete (step 350).

Other embodiments are within the following claims. For example, the above described techniques apply to all types of systems in addition to database systems, e.g. file systems, volume managers, log devices, and all hosts. The only requirement is that there be a software application for implementing the functionality on the host. It is independent of the type of software that is running or the operating system.

What is claimed is:

1. A method of generating in a tertiary storage device a backup copy of data, said method comprising:

writing data to a first set of storage volumes;

while data is being written to the first set of storage volumes, mirroring the contents of the first set of storage volumes into a second set of storage volumes; after an amount of data has been written to the first set of volumes, terminating the mirroring;

while mirroring is terminated, continuing to write data to the first set of volumes thereby causing the second set of volumes to become unsynchronized from the first set of volumes;

while mirroring is terminated, generating archived redo logs for transactions that are applied to data stored in the first set of storage volumes;

while the mirroring is terminated, backing up data stored in the second set of volumes to the tertiary storage device;

after backing up the second set of volumes, resuming mirroring of the first set of volumes to the second set of volumes;

after backing up the second set of volumes, transferring the archived redo logs to the tertiary storage device; storing the archived redo logs in the tertiary storage device along with the backed up data that was stored in the tertiary storage device; and

after resuming mirroring, incrementally resynchronizing the first and set sets of volumes with each other.

2. The method of claim 1, further comprising:

providing a single data storage system with a plurality of data storage volumes; and

partitioning the plurality of data storage volumes into the first and second sets of storage volumes.

3. The method of claim 1, further comprising:

providing a first data storage system;

providing a second data storage system that is separate from the first data storage system;

directly connecting the first and second data storage systems through a communication link;

establishing the first set of storage volumes within the first data storage system; and

establishing the second set of storage volumes within the second data storage system, and

wherein the step of incrementally resynchronizing the first and second sets of storage volumes takes place over the communication link.

4. The method of claim 1 further comprising:

defining a first map of the first set of storage volumes, said first map providing a mechanism for identifying which portions of the first set of storage volumes become out of synch with counterparts in the second set of storage volumes due to modifications which occur to the first set of storage volumes while mirroring is terminated;

defining a second map of the second set of storage volumes, said second map providing a mechanism for identifying which portions of the second set of storage volumes become out of synch with counterparts in the first set of storage volumes due to modifications which occur to the second set of storage volumes while mirroring is terminated; and wherein the step of resynchronizing the first and second sets of storage volumes comprises using the first and second maps to identify which portions of the second set of storage volumes need to be refreshed from the first set of storage volumes to achieve resynchronization.

5. The method of claim 1 further comprising providing a tape drive storage device as the tertiary storage device.

17

6. The method of claim 1 further comprising:
providing a first processor connected to the first data storage system, wherein the first processor is running a database application which is processing transactions and writing data from those transactions to the first set of storage volumes;
providing a tape drive storage device as the tertiary storage device; and
providing a second processor connected to the second data storage system, wherein tape drive storage device is connected to the second processor which performs the backing up of data in the second set of volumes to the tape drive storage device.
7. The method of claim 4 wherein the first map comprises a first set of submaps equal in number to the number of volumes in first set of storage volumes and each of which corresponds to a different one of the volumes in the first set of storage volumes, wherein the second map comprises a second set of submaps equal in number to the number of volumes in second set of storage volumes and each of which corresponds to a different one of the volumes in the second set of storage volumes.
8. The method of claim 7 wherein each of the volumes of the first and second sets of volumes is made up of a corresponding set of tracks, and wherein each of the submaps of the first and second sets of submaps has a number of records that is equal to the number of tracks in the set of tracks of the volume corresponding to that submap.
9. The method of claim 4 further comprising:
generating a first ordered list of identifiers of records within the combined map that have changed;
generating a second ordered list of identifiers of records within the second map that have been changed; and
using the first and second generated lists of identifiers to perform incremental resynchronization.
10. A method of generating a backup of data, said method comprising:
providing a first data storage system, a second data storage system, and a communications path connecting the first data storage system to the second data storage system;
linking a first region of storage in the first data storage system to a corresponding second region of storage in

18

- the second data storage system so that data written to the first region of the first data storage system is automatically written to the corresponding second region in the second data storage system;
- generating in the first data storage system a first map of said first region of storage in the first data storage system, said first map providing a mechanism for identifying which portions of said first region of storage become out of synch with counterparts in the second region storage due to modifications which occur to the first region of storage while unlinked from the second region of storage;
- generating in the second data storage system a second map of said second region of storage in the second data storage system, said second map providing a mechanism for identifying which portions of the second region of storage become out of synch with counterparts in the first region of storage due to modifications which occur to the second region of storage while unlinked from the first region of storage;
- unlinking the first region of storage in the first data storage system from the second region of storage in the second data storage system so that data written to the first region of the first data storage system is no longer automatically written to the second region of storage in the second data storage system;
- after unlinking, continuing to write data to the first region of storage in the first data storage system;
- using the first and second maps in the first and second data storage systems to record where changes are made to the first and second regions on the first and second data storage systems, respectively;
- relinking the first region of storage in the first data storage system to the second region of storage in the second data storage system so that data written to the first region of the first data storage system is again also automatically written to the second region of storage in the second data storage system; and
- after relinking, resynchronizing the first region and the second region with each other, said resynchronizing step using the first and second maps.

* * * * *



US005890165A

United States Patent [19]

Bondrie et al.

[11] Patent Number: 5,890,165

[45] Date of Patent: Mar. 30, 1999

[54] METHOD AND APPARATUS FOR AUTOMATIC DISCOVERY OF DATABASES

[75] Inventors: Robert A. Bondrie, Natick; Madhav G. Mutalik, Northboro, both of Mass.

[73] Assignee: EMC Corporation, Hopkinton, Mass.

[21] Appl. No.: 625,014

[22] Filed: Mar. 29, 1996

[51] Int. Cl.⁶ G06F 17/30

[52] U.S. Cl. 707/202; 395/182.18; 395/200.64; 707/204; 707/205

[58] Field of Search 395/67, 500, 182.18, 395/200.64, 202, 204, 205; 364/282.1, 182.04, 283.1, 222.81, 222.82, 283.4, 268.2; 707/1, 10, 101, 204, 200, 201, 205

[56] References Cited

U.S. PATENT DOCUMENTS

4,714,995	12/1987	Hatana et al.	364/200
5,109,515	4/1992	Laggis et al.	395/725
5,185,860	2/1993	Wu	395/200
5,276,872	1/1994	Lomet et al.	395/600
5,561,797	10/1996	Gilles et al.	395/600
5,594,900	1/1997	Cohn et al.	707/204
5,602,936	2/1997	Green et al.	382/104
5,644,698	7/1997	Cannon	395/182.04
5,673,382	9/1997	Cannon et al.	395/182.04
5,675,725	10/1997	Malcom	395/182.04
5,687,367	11/1997	Dockter et al.	707/102

OTHER PUBLICATIONS

Mosse et al. "Analysis of a Fault-Tolerance Multiprocessor Scheduling Algorithm", pp. 16-25, IEEE, Jan. 1994.

King et al, Gervicus of disaster recovery for transaction processing systems, pp. 286-293, IEEE, Jul. 1990.

Primary Examiner—Thomas G. Black

Assistant Examiner—Jean M. Corriellus

Attorney, Agent, or Firm—Maureen Stretch

[57]

ABSTRACT

A method and apparatus for automatic discovery of databases that determines what databases are installed at a client site, interrogates each database to determine what files or tables are present and what storage is associated with each, groups the databases into the smallest atomic units that can be backed up while preserving database integrity, and presents this information to backup procedures. In a preferred embodiment, a discovery process is performed by a script for each client to be backed up. Each script causes the creation of a temporary file describing each database. The temporary files are normalized into a common format which is then passed to a grouper program that analyzes the temporary files and produces a data descriptor file describing the smallest atomic units that can be backed up while still preserving database integrity at that client site. This data descriptor file is presented to a work item generator for scheduling backup procedures. Backups of each atomic unit can then be scheduled as desired.

6 Claims, 14 Drawing Sheets

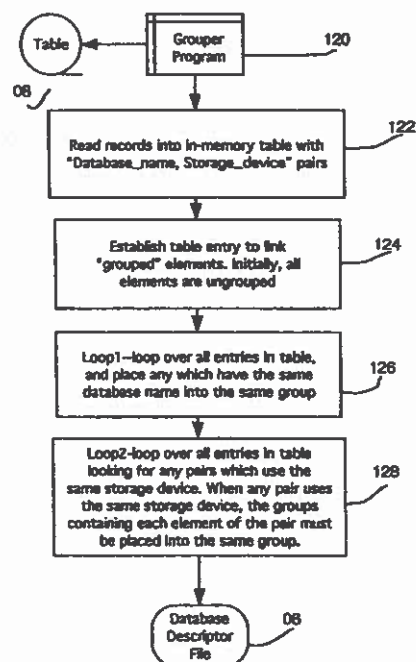
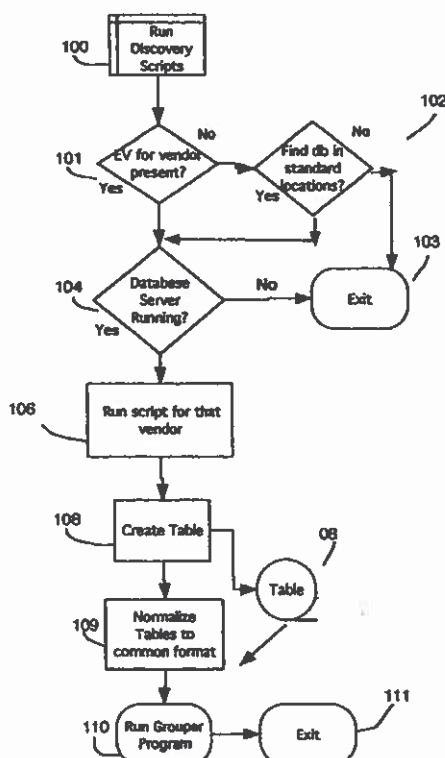


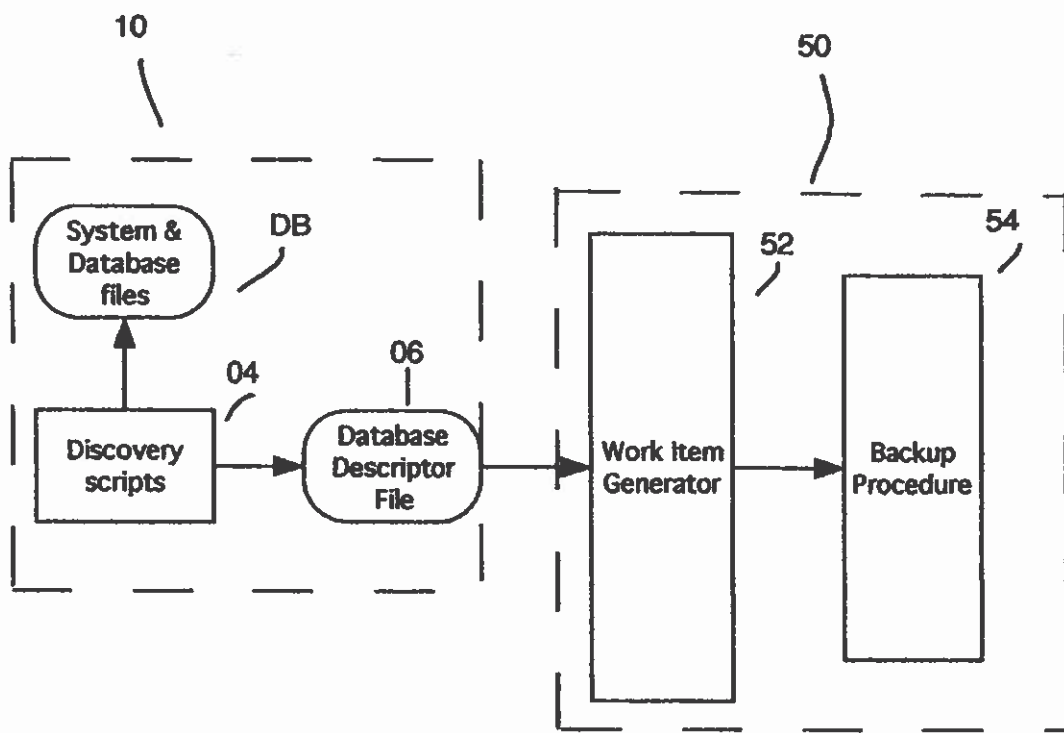
Fig. 1a

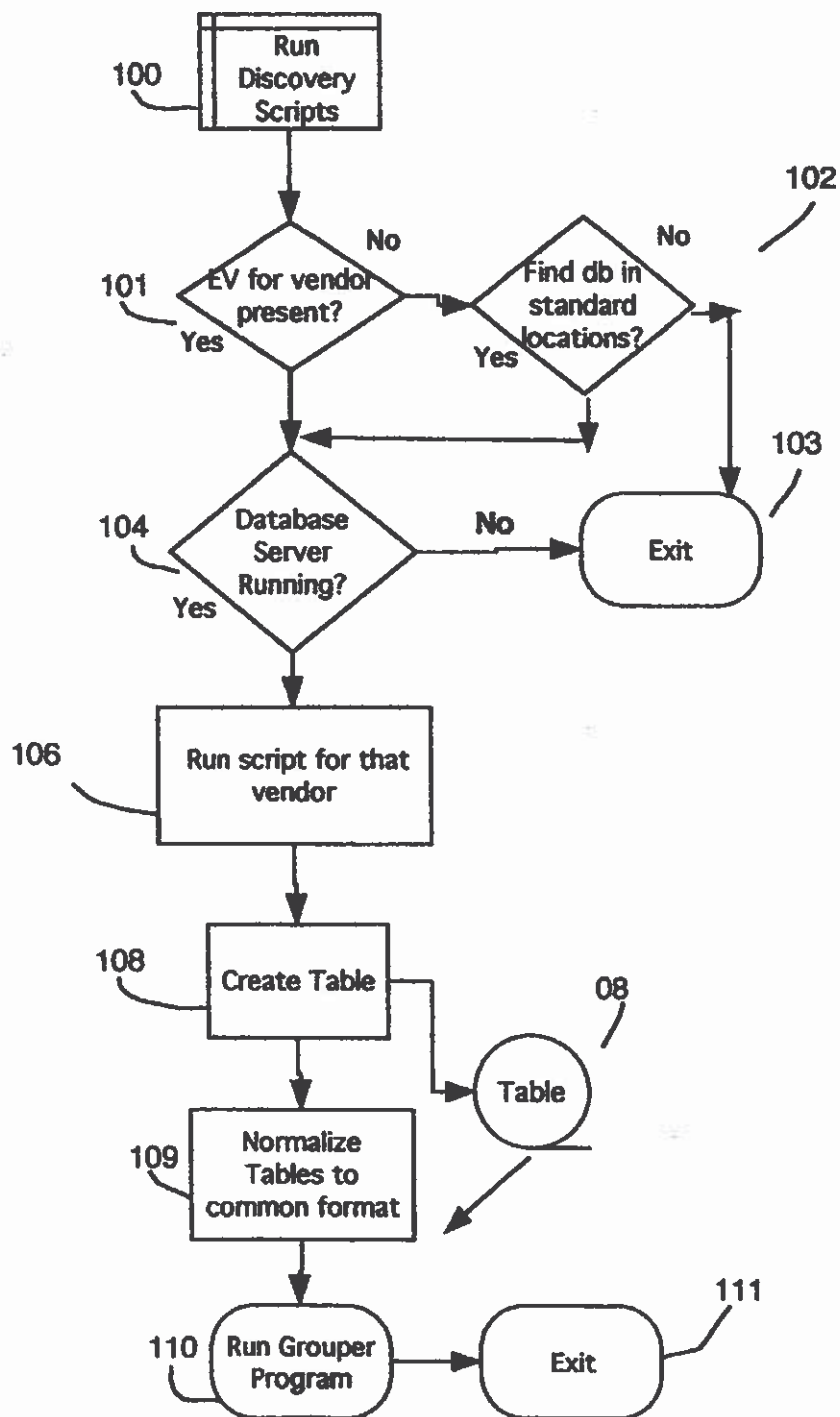
Fig. 1b

Fig. 1c

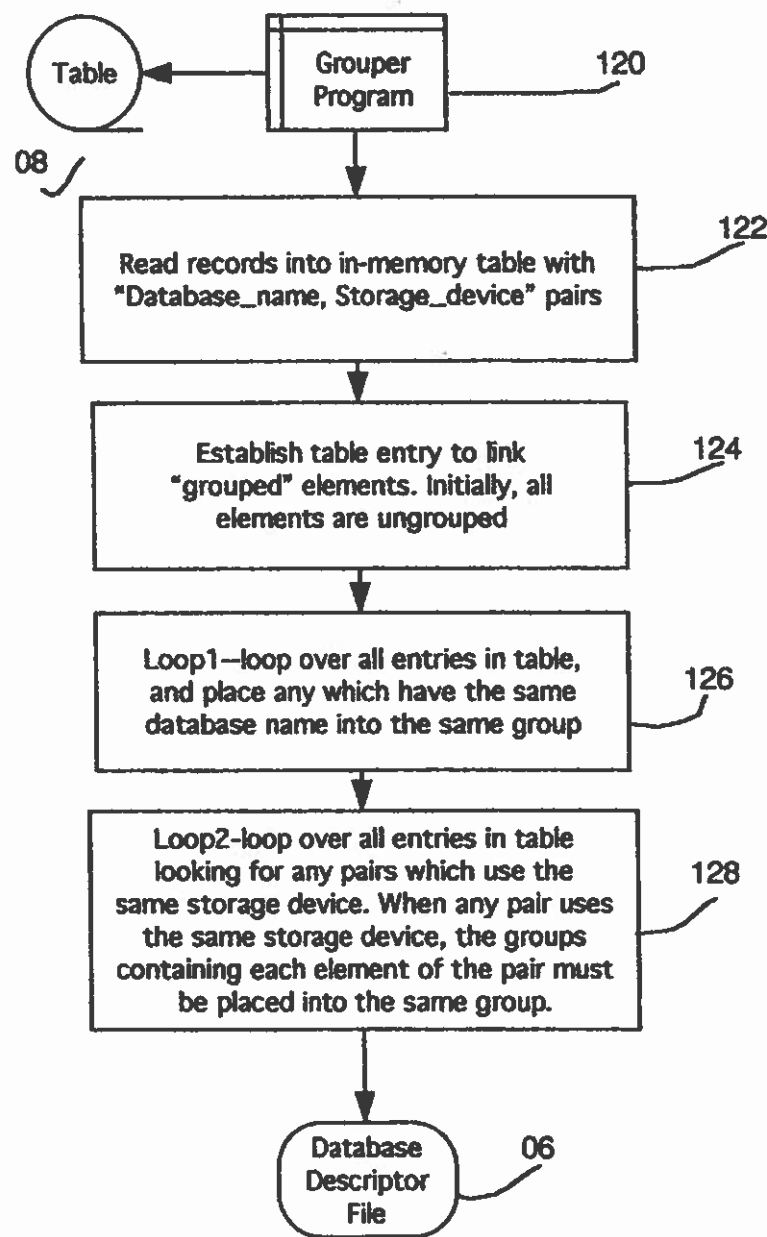


Fig. 1d

database_name	storage_device	group
DBA	S1	1
DBA	S2	1
DBA	S3	1
DBB	S4	2
DBB	S5	2
DBC	S6	3
DBC	S7	3
DBD	S7	3
DBD	S7	3

Fig. 2

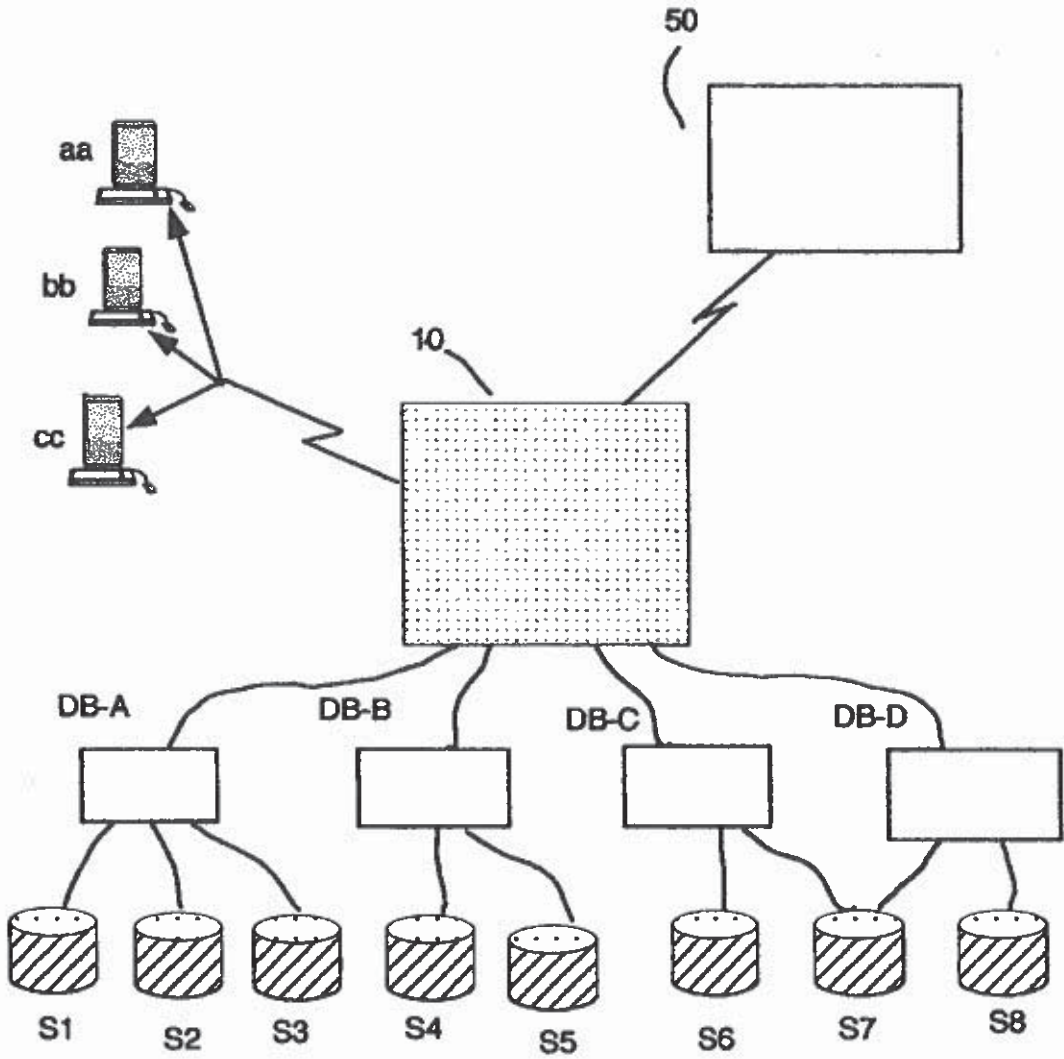


Fig. 3a

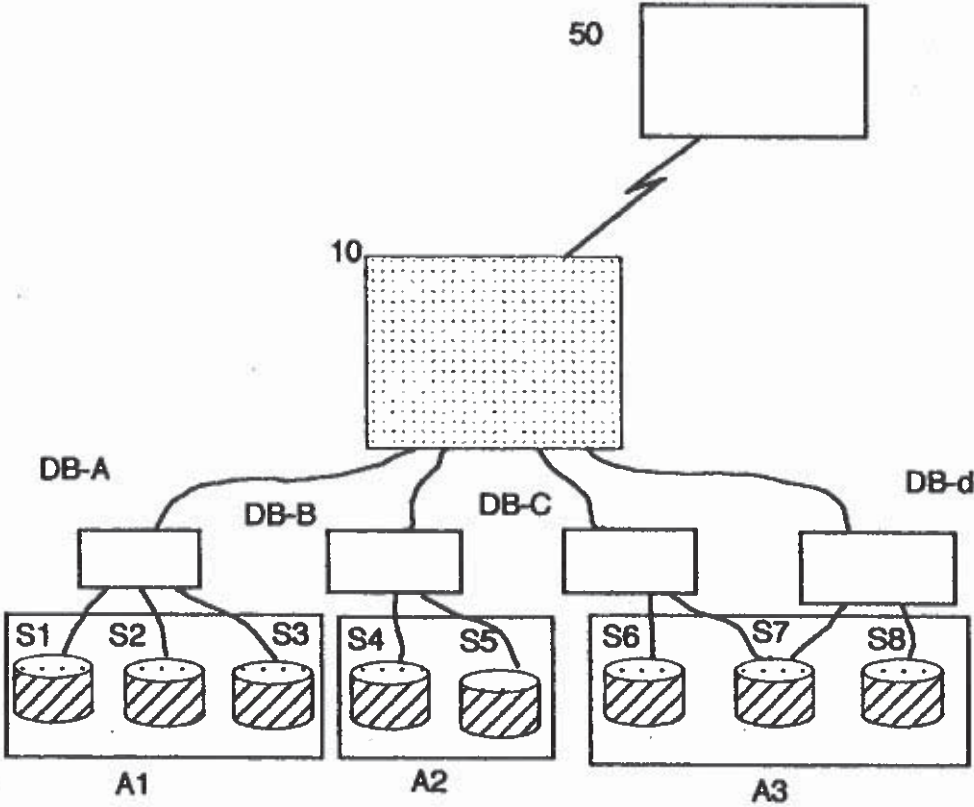


Fig. 3b

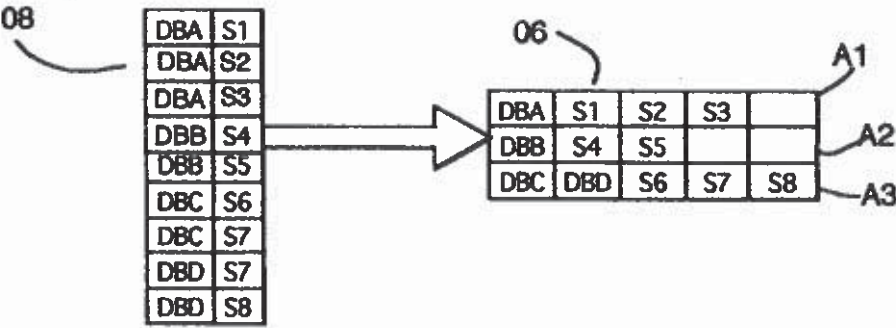


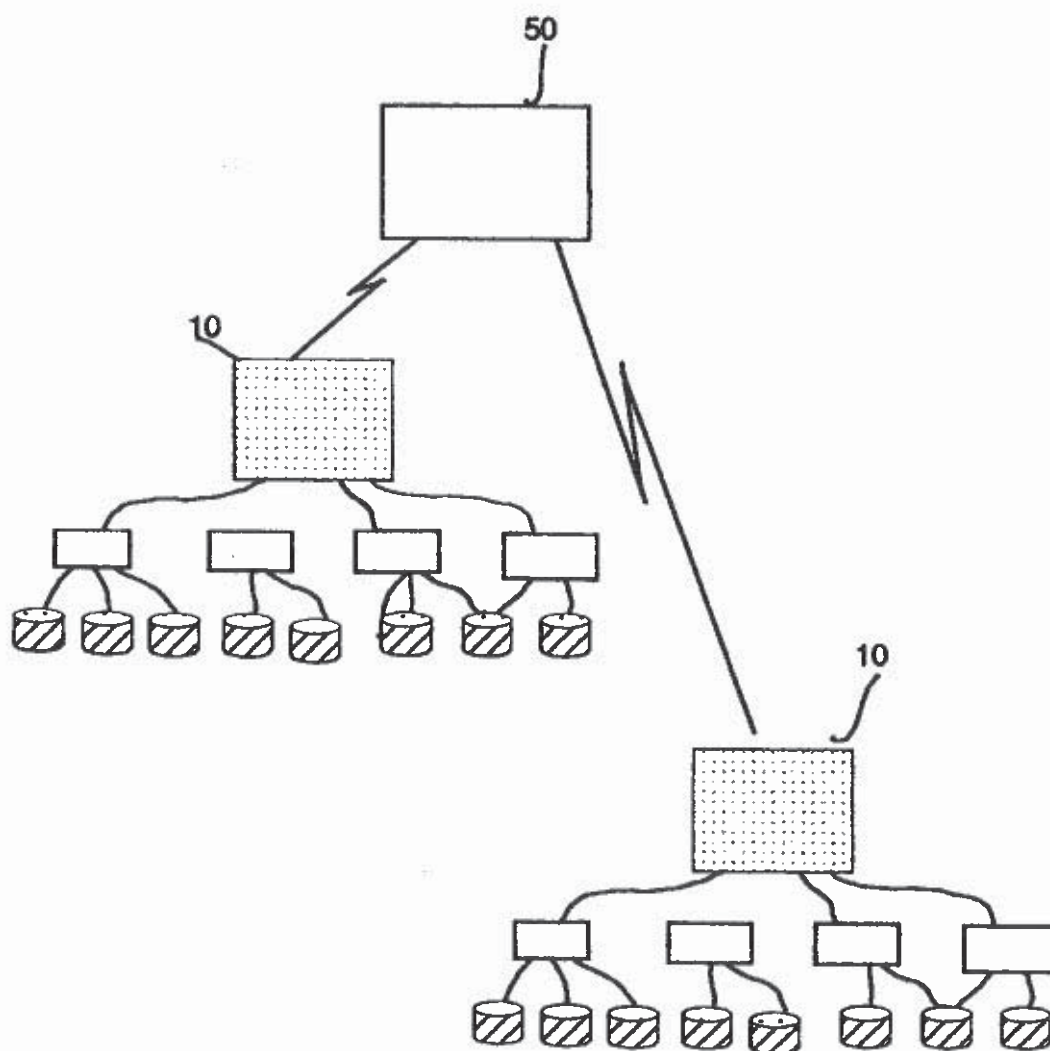
Fig. 4

Fig. 5.

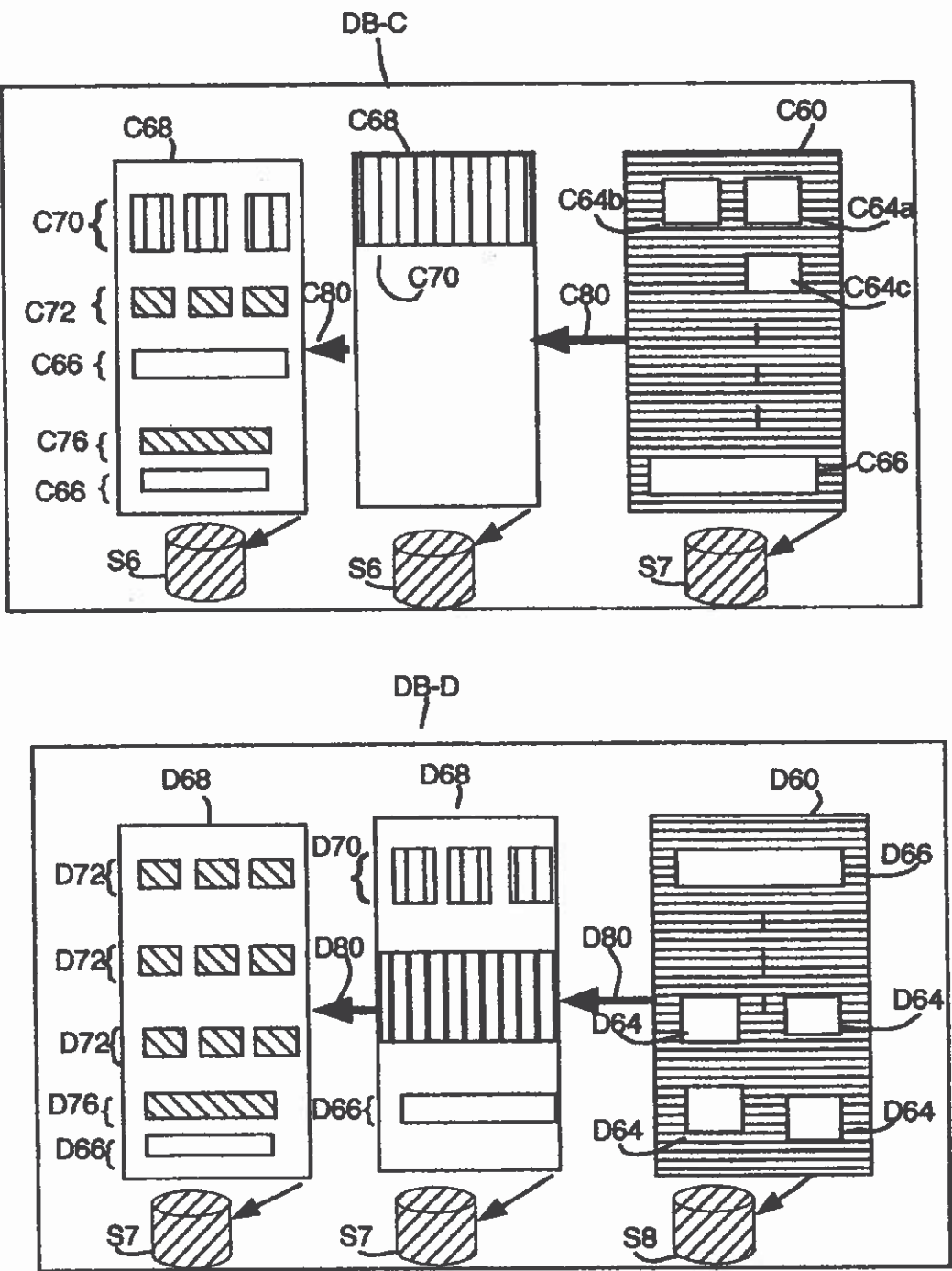


Fig. 6

Database	Environment variable which must be set	V1
Informix	INFORMIX_DIR	V2
Oracle	ORACLE_HOME	
Sybase	SYBASE	V3

Fig. 7b

```
cent_db, \
J,4,1,1,5,cname,5,sname,3,5,dbrev,0x3,4,8,cent2_db,7,
ive15.log,2, \
4,rob1,4,rob2,2,19,/SYBASE/Ldrive3.log,20,/SYBASE/Ldr
18,/dev/rdisk/c201d2s0,18,/dev/rdisk/c201d4s0,0;
```

Which translates to:

```
Pcode      : 1          /* Platform code
Client     : cname      /* Client name
Server     : sname      /* Server name
Dbase type : 3          /* Database type code num
Dbase rev  : dbrev      /* Database revision
Dbase flag : 0x0000     /* Database info flags
Num databi: 4          /* Num dbases in group
Databi1:   /* Names of databases
cent2_db
cent_db
rob1
rob2
Num files: 2          /* Num files used to backup
/SYBASE/Ldrive3.log /* data, followed by file
/SYBASE/Ldrive15.log /* names.
Num devices: 2        /* Num physical devices
/dev/rdisk/c201d2s0 /* used for backup, followed
/dev/rdisk/c201d4s0 /* by device names
```

Fig. 8

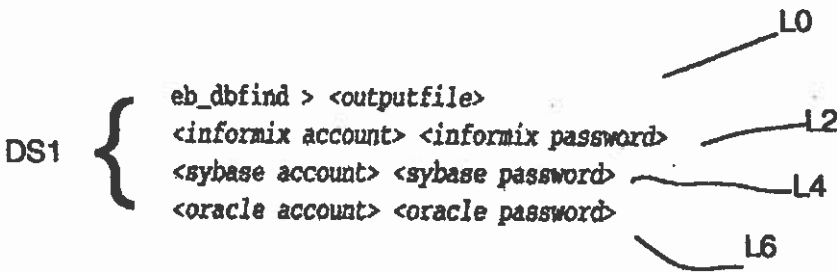


Fig. 9

Database	Dbase Password(s) required	File/Directory Access Requires
Informix		
Oracle	Userid, password capable of reading v\$logfile, v\$datafile and v\$controlfile views.	/etc/oratab
Sybase	"sa" userid, password, or userid/password of acct. w/sa priv	\$SYBASE/install

Fig. 10a

```
sql> connect sys/sys_password;  
create user eadmin identified by password;  
grant select on v_$logfile to eadmin ;  
grant select on v_$datafile to eadmin;  
grant select on v_$controlfile to eadmin;  
grant create session to eadmin;
```

Fig. 10b

```
isql -Usa -Psa_password  
sp_addlogin eadmin,password,master  
go  
sp_adduser eadmin,eadmin  
go  
grant select on sysdevices to eadmin  
go  
grant select on sysusages to eadmin  
go  
grant select on sysdatabases to eadmin  
go
```

METHOD AND APPARATUS FOR AUTOMATIC DISCOVERY OF DATABASES

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates generally to the field of backup and recovery of databases and more particularly to the automatic discovery of databases on storage devices so that they can be grouped for backup purposes.

2. Background

Companies and institutions that use databases in their computer systems often have several of them, some of which may be created and managed by software from different vendors. For example, a large multinational manufacturing company may have at one client site a relational database for handling its manufacturing and inventory data. That company's sales department might use a separate relational database for tracking sales and accounts at that same client site. The company's finance department might use still another relational database to track assets and finances at that client site. It is possible that these databases will be stored on the same storage systems at the client site, which can, in turn, be accessed by the same servers or mainframe computers.

In the example given, the inventory data base may be created and managed by software supplied by Oracle™ Corporation, while the sales database may be created and managed by software from Informix™ Corporation or Sybase™ Corporation, or another supplier of relational databases. While such a mixing of software from different vendors is not the most common usage, it does occur with some frequency.

Since most of these databases hold critical information, regular and reliable backups are important. In addition to backups done for each database, most systems administrators consider it prudent to have disaster recovery backups available as well.

Heretofore, systems or database administrators who wished to perform the backups of such databases had to determine what storage units were used by which databases, which type of database used each, and how the different database systems allocated data to the storage units.

Such an effort is complicated by the fact that one of the most common types of database is a relational database and each relational data base has a number of files or tables created for it by the database application program. One Oracle™ relational database, for example, may have control files, data files, online redo log files, archive redo log files, initialization parameter files, and a file containing the Oracle™ code, among other files. Traditional backup planning requires that the system or database administrator know where each of these files for a given database is located on the storage units available to the system or server. FIG. 5 illustrates how a Sybase™-like database, for example, DB-C, might allocate indexes C66, catalog tables C64, data tables C70, C72, into two logical groupings, system tables C60 and user database tables C68. In turn, these logical groupings may be physically allocated to two different storage units S6 and S7.

A vendor may supply its own backup and restore programs or offer guidelines for using commonly available operating system software utilities to perform backups and restores. Whether unique programs are supplied or only guidelines, these differ from each other and cannot be used interchangeably. That is, the Informix™ backup and restore

procedures usually will not work with a database from Sybase™. Since each database vendor also allocates data to storage units differently, system-wide backup and restore efforts require a significant level of expertise on the part of the administrator(s).

Space for each database at a client site may be grouped across several storage devices, as shown in FIG. 2. If two databases from different vendors are allocated to the same storage unit at one client site, and that unit is backed up, all the databases that have any files on that device must be completely restored to preserve database integrity. To illustrate, suppose database DB-C of FIG. 2 is allocated across storage units S6 and S7 and database DB-D is allocated across units S7 and S8. Databases DB-C and DB-D share a storage unit S7. Assume a backup copy is made of storage of units S6, and S7 for database DB-C, using an operating system-level utility. After the backup copy is made, database DB-D is updated by many transactions, but database DB-C remains as it was. If a failure occurs and the backup copy is restored to units S6 and S7, database DB-C is operational again, but database DB-D may be corrupted. This problem can occur even if both databases DB-C and DB-D use the same vendor software.

If a given installation attempts to eliminate the risk of corruption by backing up each database individually, under manual control, using the procedures or programs designated by the vendor, that, too, can be problematic since it is slow and is prone to manual mistakes.

Consequently, if client site complete backups are done using present methods they are usually done either by writing all storage units to backup devices at one time, which restricts scheduling flexibility and requires that all units be restored at the same time, as well, or by requiring that each individual database be backed up by an expert administrator using the tools or system procedures supplied for use with only that database.

It is an object of the present invention to provide a way to discover and backup databases that is more convenient for the user than present methods.

Yet another object of the present invention is providing a method and apparatus to discover and backup databases that requires less user level expertise than present methods.

Still another object of the present invention is providing a method and apparatus to discover and backup databases that is less error prone than a primarily manual process.

SUMMARY OF THE INVENTION

These and other objects of the invention are achieved by a method and apparatus for automatic discovery of databases that determines what databases are installed at a client site, interrogates each database to determine what files or tables are present and what storage is associated with each, groups the databases into the smallest atomic units that can be backed up while preserving database integrity, and presents this information to backup procedures. In a preferred embodiment, a discovery process is performed by a script for each client to be backed up. Each script causes the creation of a temporary file describing each database. The temporary files are normalized into a common format which is then passed to a grouper program that analyzes the temporary files and produces a data descriptor file describing the smallest atomic units that can be backed up while still preserving database integrity at that client site. This data descriptor file is presented to a work item generator for scheduling backup procedures. Backups of each atomic unit can then be scheduled as desired.

It is an aspect of the present invention that it performs automatic discovery of databases.

Another aspect of the present invention is that it provides additional scheduling flexibility to the backup and restore of databases without compromising data integrity.

It is a feature of the present invention that by grouping databases into atomic units the invention reduces the likelihood of compromising data integrity.

Yet another feature of the present invention is that it reduces the likelihood of errors arising through manual procedures.

Still another aspect of the present invention is that it is simple to use in connection with databases supplied by different vendors.

BRIEF DESCRIPTION OF THE DRAWINGS

- FIG. 1a is a block diagram of the present invention.
- FIG. 1b is a flow diagram of the discovery process of the present invention.
- FIG. 1c is a flow diagram of a grouper program according to the method and apparatus of the present invention.
- FIG. 1d is a table illustrative of the results of the grouper program according to the method and apparatus of the present invention.
- FIG. 2 is a schematic diagram of databases allocated to storage units on a client accessed by a server.
- FIG. 3a is schematic diagram of a grouping of databases according to the method and apparatus of the present invention into atomic units.
- FIG. 3b is set of tables illustrating the grouping of databases according to the method and apparatus of the present invention into atomic units.
- FIG. 4 is a schematic diagram of several database clients accessed by a server.
- FIG. 5 is a schematic diagram of relational database tables and files as they might be allocated to storage units.
- FIG. 6 is a table of environment variables that must be set according to the method and apparatus of the present invention in a preferred embodiment.
- FIG. 7 is a block diagram of an database descriptor file according to a preferred embodiment of the present invention.
- FIG. 7b is a partial list of field definitions in a database descriptor file according to a preferred embodiment of the present invention.
- FIG. 8 is a schematic diagram of a script for the automatic discovery process of the present invention.
- FIG. 9 is a table showing items of information required by various database vendors for access.
- FIG. 10a is a listing of a partial discovery script for an Oracle™ database according to the method and apparatus of the present invention.
- FIG. 10b is a listing of a partial discovery script for an Informix™ database according to the method and apparatus of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

In FIG. 1, a schematic flow diagram of the present invention is shown. In a preferred embodiment, the invention operates on one or more clients 10, containing a number of system and database files DB. Discovery scripts are performed on client 10 to create a database descriptor file 06

describing the smallest atomic units of system and database files that can be backed up without compromising data integrity. Database descriptor file 06, in turn, can be accessed by a work item generator 52, operating on a storage management server 50 that is in communication with client 10. Work item generator 52 communicates with the backup procedures 54 used by the server to backup the system and database atomic units listed in database descriptor file 06.

In FIG. 2, an example of system and database files DB is shown. It can be seen that storage management server 50 is in communication, either through local connection or over a network, with client 10. Client 10, in turn, acts as a database server to personal computers aa, bb and cc in a network. In a preferred embodiment, storage management server 50 is usually a mainframe computer or powerful workstation computer running a storage management system, such as EMC Corporation's Epoch Database Management System. Depending on the user's installation, client 10 could be a workstation computer, a mainframe computer, a minicomputer or a personal computer. As will be apparent to those skilled in the art, client 10 and storage management server 50 could also be the same computer, in which case the client/server analogy is less apt.

Still in FIG. 2, at client 10, there are four databases DB: DB-A, DB-B, DB-C and DB-D. Each database is allocated to specific storage devices by the vendor-supplied program that created the database, such as programs from Oracle™, Informix™, Sybase™, etc.

Database DB-A, for example, is allocated to space on three storage units S: S1, S2, and S3. Database DB-B has been allocated to two more storage units S: S4 and S5. Database DB-C has been allocated to storage units S6 and S7. Note that the fourth database, DB-D, in this example, has been allocated to storage units S7 and S8. In this example, databases DB-C and DB-D share storage unit S7.

Most general purpose backup programs operate most efficiently by copying all the data on one or more designated storage units to some backup medium, such as tape or another storage device. Thus, a normal backup of database DB-C would copy storage units S6 and S7 in their entirety to the backup media. If this media is used to restore database DB-C a day after the backup has been made, storage units S6 and S7 will be returned to the state they were in when the backup was made. Any changes that have been made to DB-D on S7 during that one-day interval will be lost. In addition, it is possible that this kind of backup and restore may cause corruption in the system and other files of database DB-D.

Thus, in the example shown in FIG. 2, neither database DB-C nor database DB-D can be backed up by itself without causing potential corruption problems for the other database. Therefore, the smallest atomic units that can be backed up as single entities in the examples shown in FIG. 2 are database DB-A, database DB-B, and the combination of databases DB-C and DB-D together.

This is shown more clearly in FIGS. 3a and 3b. In FIG. 3a storage management server 50 and client 10 are shown again having the same database configuration as that depicted in FIG. 2. However, in FIG. 3a, the smallest atomic units A, that can be backed up individually are shown as A1, A2 and A3. Atomic unit A1 contains only database DB-A, since no other databases share any of the storage units S1, S2 or S3 allocated to it. The same is true for database DB-B. It is atomic unit A2, since no other database is allocated to the units used by it. However, databases DB-C and DB-D, together, comprise atomic unit A3, because they do share

5

storage unit S7. Since they share this unit, any general level backup or restore of the entire unit affects both databases.

Referring now to FIG. 3b, the overall logic of the grouping of databases into atomic units is shown. Data table 08 is a schematic representation of a normalized temporary file created by using scripts to identify which databases are present on client 10 and the storage units Sn, where n is the identity of a unit, to which they are allocated.

Referring now to FIG. 1b, the overall logic of the present invention is shown. In a preferred embodiment, the database server program must be running on client 10. A script is run to create a temporary file. In a preferred embodiment, a grouper program can then be executed to analyze table 08 and produce database descriptor file 06, which lists the atomic units contained in table 08. As shown in more detail in FIG. 1b, at step 100, a particular type of database is selected by looking for its environment variable in operating system directories. In a preferred embodiment, the Unix operating system is used, but as will be apparent to those skilled in the art, any of a number of operating systems which permit the user of scripts or stored command sequences can be used.

If the environment variable for a specific vendor is present in the system directories of the operating system running on client 10, an attempt to run the script on that vendor's server on client 10 is made at step 104. If the server is running, the script will execute. If the server is not running, no discovery is performed, in a preferred embodiment.

If the server is running, the script for that vendor is executed at step 106. For relational databases, the script is an SQL script which that vendor's software will execute to create a temporary file indicating where all the files are located. For other types of databases, either a script or an executable program or utility can be run to locate the database and create the temporary file.

Once a temporary file has been created for one data base, steps 104 through 108 are executed for each database vendor software type identified in the system environment variables. If 6 databases exist, 3 created by Oracle™, 1 by Sybase™ and 2 by Informix™, then 3 temporary files will be created, one for each vendor. Next, at step 109, these three files, which, in a preferred embodiment, are in ASCII format produced by the various database vendors, are normalized by a Unix or shell script that takes the three ASCII files in, and creates one output ASCII temporary file that is in uniform format. That file is generally in the format illustrated in data table 08 in FIG. 3b. That file becomes the input to a grouper program according to the method and apparatus of the present invention at step 110.

Turning now to FIG. 1c, a flow diagram of a grouper program according to the method and apparatus of the present invention is presented. At entry to grouper program 120, table 08 is provided as input. At step 122, grouper program 120, reads records from table 08 into an in-memory table with columns named

"Database_name" and "Storage_device" as shown in FIG. 1d. At step 124, a table entry named group is established to link "grouped" elements. Initially, all the elements are ungrouped. Next, at step 126, a first loop is performed, iterating over all entries in the table and placing any which have the same database_name into the same group. Still in FIG. 1c, next, at step 128, a second loop iterates over all entries in the table looking for any pairs which use the same storage device. When any pair uses the same storage device, the groups containing each element of the pair must be placed into the same group.

6

When steps 126 and 128 are completed the result is the database descriptor file 06, which is then written to storage in a preferred embodiment. As will be apparent to those skilled in the art, however, it could also be sent as a message or communication file to work item generator 52.

Now with reference to FIG. 6, a data descriptor file 06 is shown in block diagram form. In a preferred embodiment, the file layout is compatible with EMC Corporations Epoch Storage Management System file formats, and in particular the format known as J,4. As will be apparent to those skilled in the art, any of a number of flat file formats, keyword formats, or other formats can be used to convey the information produced by the grouper program to a suitable backup scheduling program or backup program.

In the format shown in FIG. 7a and 7b, the J,4 record descriptor appears first in the file. This is followed by a subtype field which is a reserved, unused field in a preferred embodiment. Next, a platform code field can be used to identify the machine type the discovery code runs on, if more than one type of computer is used. Next a database name appears. For each atomic unit An, a J,4 type record is created.

Returning to FIG. 1a, once a database descriptor file 06 describing the atomic units found in system and database files DB is created, it can be sent or accessed by a work item generator 52 running as part of an overall storage management system on storage management server 50. In a preferred embodiment, this is EMC Corporation's Epoch Storage management system. In that system, a program known as a work item generator reads files to determine what work is to be scheduled for the backup and recovery programs of the storage management system. Work item generator 52 is then able to schedule backups for the various atomic units An, and convey that information to the backup procedure 54. This provides a significant level of flexibility to the installations represented by storage management server 50 and client 10. Rather than require system administrators to become expert in each different vendor database in order to schedule and execute regular backups of them, the discovery scripts 04 collect the information about the system and database files DB for analysis by the grouper program to generate database descriptor file 06. In turn, work item generator 52 at the storage management server 50, can then schedule backups for each atomic unit An, identified in descriptor file 06. The schedules selected can be defined and customized for each client/server to optimize efficiencies.

Now turning to FIG. 4, it can be seen that several clients 10 may be managed in this way by the storage management processes on storage management server 50. In the past, as this type of expansion occurred, without the present invention, more training and expertise needed to be developed in the organization to manage this. The present invention makes it easier to accommodate and manage this kind of growth.

In the same way, in the past, if one wanted to have full unit backups made regularly of all databases, the most prudent method would have been to shut down or quiesce all systems and activities while the backups were done all at one time. According to the method and apparatus of the present invention, the benefit of regular full backups can be obtained without shutting down all systems and activities. Instead, backups can be scheduled by storage management server 50 to occur in an orderly fashion for the various atomic units An spread throughout each client 10. Shutdown or quiescing of the database server for a particular database vendor on a client 10 need only be done when an atomic unit An is being

backed up. In some cases, only the instance of that database needs to be quiesced on that server.

As mentioned earlier, the complexity of individual relational databases is illustrated in FIG. 5. The two different relational databases DB-C and DB-D shown earlier in FIG. 3 are schematically depicted in FIG. 5 in more detail, both as they might be structured logically and as they might be physically allocated to storage. Database DB-C, for example, has a system table C60, that contains an index C66, and several catalog tables C64a-c. This system table C60 is shown as physically allocated to storage unit S7. Addresses and other connectors C80 logically connect the system table C60 to user data C68. Each collection of user data C68, in turn may have a number of data tables C70 and C72, and C76 as well as indexes C66. In this example, user data C68 is shown allocated to storage unit S6 for database DB-C.

In the same FIG. 5, a second database, DB-D is shown, also having system tables D60, and user data D68. In this database DB-D, the location and the number of catalog tables D64, as well as data tables D72, D70 and D76 are different, as would be their contents. In this example, user data D68 for database DB-D is physically allocated to storage unit S7, while system tables D60 are allocated to storage unit S8.

The database management programs that create and manage such tables and structures as those shown in FIG. 5 vary in the way they organize the tables and files both logically and physically. The present invention backs up all the storage units associated with an atomic unit, going directly to the device or "raw" file, as it is sometimes called, itself, thus bypassing the database access in order to gain speed.

Different vendors also have different access requirements for databases created by their respective programs. For example, as shown in FIG. 9, databases created by the Oracle™ database management program require a Userid and password that is designated as having the right to access such things as the Oracle™ v\$logfile, v\$datafile and v\$controlfile views of the databases. (Most relational databases permit users to create "views" of the data. Views may sometimes be called virtual tables or derived tables.)

Still in FIG. 5, when a particular vendor's database program is installed on a computer system such as client 10, various directories and files are created on the main directories of the operating system software that is executing on client 10. In a preferred embodiment, it is assumed that a version of the Unix operating system is running on the cpu in client 10. Thus, when database software is installed, the main directories of the operating system on client 10 are updated to include references to this installation.

As indicated in FIG. 6, if the Informix™ database software has been installed on the client 10, there will be an environment variable (EV) Vn,, present such as directory entry V1, indicating that Informix™ is installed. Similarly, for Oracle™, and Oracle™_Home entry V2 will exist. And for Sybase™, a directory entry V3 will be found. In a preferred embodiment, discovery scripts DS1, always check for an environment variable to see if the database is installed. If the environment variable is not set, the present invention will look for the database in some standard locations. For example, it will look for an "oratab" file in the system area for Oracle™ databases, or look for an Informix™ account password file, for Informix™ databases, or for Sybase™ databases, it will look for a /SYBASE™ directory or link. These kinds of indicators of the existence of a database are usually known for each type of database vendor and operating system environment.

FIG. 8 shows a discovery script DS1 used according to the method and apparatus of the present invention. In a preferred embodiment, a line is created in a discovery script for each database vendor, such as the discovery script DS1 shown in FIG. 8. There, line L0 of discovery script DS1, refers to a set of discovery scripts called dbfind, which will create a data descriptor file 06, called <outputfile> in this example. Assuming databases from three vendors, Informix™, Sybase™ and Oracle™ are present on client 10, a system administration account has usually been created for each, by a system administrator or database administrator, having system-wide access privileges associated with its password.

When the system administration account is created, the appropriate environment variables are also associated with each vendor. Returning to FIG. 8, line L1 of DS1 names the Informix™ system administration account name and uses the system password for it, line L2 identifies the Sybase™ system administration account name and password, and line L3 names the Oracle™ system administration account and password. Thus, dbfind is a script which causes discovery scripts to be run, in this example, for each of the 3 database types. The discovery scripts are usually a combination of Unix or Unix-shell scripts and database SQL scripts. The shell scripts locate information stored in operating system files and directories, while the SQL scripts are run by the database server itself and locate information known to the vendor-specific software. As will be apparent to those skilled in the art, these scripts could also be executable programs, but using shell scripts and SQL scripts provides simpler editing and maintenance in a preferred embodiment.

In a preferred embodiment, the use of a system administration account and an associated password as shown in FIG. 8 provides the simplest way to set up a discovery script DS1. As will be apparent to those skilled in the art, this approach while very broad in scope, has the drawback of giving very broad privileged access to the account. If there are several clients 10 being managed for backup purposes by one storage management server 50, the users at one or more of the clients 10 may not want anyone to have such broad access.

In an alternative preferred embodiment, then, minimally privileged accounts are created for each vendor at a client site. In these cases, Structured Query Language (SQL) scripts such as those depicted in FIGS. 10a and 10b are used. FIG. 10a shows the script used for an Oracle™ database, FIG. 10b shows the one used for a Sybase™ database.

As will be apparent to those skilled in the art, rather than using scripts, an executable program could be written to perform the same functions. However, the use of scripts allows the invention to run on all standard Unix platforms supported by the vendors. An executable would probably have to be recompiled for each platform.

As will be apparent to those skilled in the art, while the examples shown are directed primarily to relational databases, the invention can be used with other types of databases provided that information about their allocations to storage units can be ascertained through some means that can be automated, such as scripts, command sequences, utilities or special purpose programs, among others.

Preferred embodiments of the present invention are designed to be run as scripts and programs stored on disks or tapes or other magnetic, optical or digitally readable media and brought into a computer system for execution.

Those skilled in the art will appreciate that the embodiments described above are illustrative only, and that other systems in the spirit of the teachings herein fall within the scope of the invention.

What is claimed is:

1. An apparatus for automatically discovering databases allocated to storage systems, comprising:

- a storage management server computer communicating with a database client computer, the storage management server computer having a work item generator and a backup program;
- a set of instructions capable of collecting file allocation information from the database client computer into a temporary file;
- a grouper program capable of reading the temporary file and analyzing it to identify as atomic units the smallest set of databases that can be backed up as a unit without compromising data integrity; and
- an output descriptor file for storing the results produced by the grouper program, so that the work item generator can schedule backups for the storage management server computer to execute, thereby providing a database discovery and backup system that does not compromise data integrity.

2. The apparatus of claim 1, wherein the set of instructions further comprises scripts stored in a known location in the storage management server computer.

3. The apparatus of claim 1, wherein the set of instructions further comprises an executable program stored in a known location in the storage management server computer.

4. A method for automatically discovering databases allocated to storage systems, comprising the steps of:

establishing communication between a storage management server computer and a database client computer, the storage management server computer having a work item generator and a backup program;

performing a set of instructions capable of collecting file allocation information from the database client computer into a temporary file;

executing a grouper program capable of reading the temporary file and analyzing it to identify as atomic units the smallest set of databases that can be backed up as a unit without compromising data integrity; and

creating an output descriptor file for storing the results produced by the grouper program, so that the work item generator can schedule backups for the storage management server computer to execute, thereby providing a database discovery and backup system that does not compromise data integrity.

5. The method of claim 4, wherein the step of performing a set of instructions further comprises the step of performing scripts stored in a known location in the storage management server computer.

6. The method of claim 4, wherein the step of performing a set of instructions further comprises the step of running an executable program stored in a known location in the storage management server computer.

* * * * *



IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Ganesh, et al

Confirmation No.: 6266

Application No.: 09/774,839

Examiner: Alford W. Kindred

Filing Date: Jan 31, 2001

Group Art Unit: 2172

Title: RECREATION OF ARCHIVES AT A DISASTER RECOVERY SITE

COMMISSIONER FOR PATENTS
Washington, D.C. 20231

RECEIVED

APR 30 2003

Technology Center 2100

TRANSMITTAL LETTER FOR RESPONSE/AMENDMENT

Sir:

Transmitted herewith is/are the following in the above-identified application:

- (X) Response/Amendment () Petition to extend time to respond
() New fee as calculated below () Supplemental Declaration
() No additional fee (Address envelope to "Box Non-Fee Amendments")
() Other: (fee \$)

CLAIMS AS AMENDED BY OTHER THAN A SMALL ENTITY						
(1) FOR	(2) CLAIMS REMAINING AFTER AMENDMENT	(3) NUMBER EXTRA	(4) HIGHEST NUMBER PREVIOUSLY PAID FOR	(5) PRESENT EXTRA	(6) RATE	(7) ADDITIONAL FEES
TOTAL CLAIMS		MINUS		= 0	X \$18	\$ 0
INDEP. CLAIMS		MINUS		= 0	X \$84	\$ 0
[] FIRST PRESENTATION OF A MULTIPLE DEPENDENT CLAIM					+ \$280	\$ 0
EXTENSION FEE	1ST MONTH \$110.00	2ND MONTH \$410.00	3RD MONTH \$930.00	4TH MONTH \$1450.00		\$ 0
OTHER FEES						\$
TOTAL ADDITIONAL FEE FOR THIS AMENDMENT						\$ 0

Charge \$ 0 to Deposit Account 08-2025. At any time during the pendency of this application, please charge any fees required or credit any overpayment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees. A duplicate copy of this sheet is enclosed.

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Washington, D.C. 20231.

Date of Deposit: Apr 22, 2003

Typed Name: Steve M. Perry

Signature: Steve M. Perry

Respectfully submitted,

Ganesh, et al

By Steve M. Perry
Steve M. Perry

Attorney/Agent for Applicant(s)
Reg. No. 45,357

Date: Apr 22, 2003

Rev 02/03 (TransAmd)

Telephone No.: (801) 566-6633

- Attach as First Page to Transmitted Papers -



PATENT APPLICATION No. 09/774,839
PDNO 10006924-1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICANT: Ganesh, et al

SERIAL NO.: 09/774,839

FILED: January 31, 2001

FOR: RECREATION OF ARCHIVES
AT A DISASTER RECOVERY
SITE

ART UNIT: 2172

EXAMINER: Alford W. Kindred

DOCKET: PDNO 10006924-1

AMENDMENT

RECEIVED

APR 30 2003

Technology Center 2100

Box Non-Fee Amendment
Commissioner for Patents
Washington, D.C. 20231

Sir:

In response to the Office Action mailed January 28, 2003, Applicants offer the following Amendment and request reconsideration of the above-captioned application.

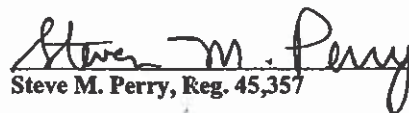
This Amendment addresses every ground of rejection set forth in the Office Action.

Please amend the application as follows:

CERTIFICATE OF DEPOSIT

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postage prepaid, in an envelope addressed to: Box Non-Fee Amendment, Commissioner for Patents, Washington, D.C. 20231, on April 22, 2003.

Respectfully submitted,


Steve M. Perry, Reg. 45,357

In the Claims:

AT
B1

1. (Currently Amended) A method for creating a standby data processing system located at a standby site, which is remotely located from a primary site, comprising the steps of:

(a) computing a difference between an active ~~data file~~transaction redo log and an archive ~~data file~~transaction redo log to produce a delta image at the primary site;

(b) transmitting the active ~~data file~~transaction redo log and the delta image, separately in time sequence, from the primary site to the standby site; and

(c) combining the delta image with the active ~~data file~~transaction redo log at the standby site, to produce a standby archive ~~data file~~transaction redo log.

2. (Cancelled.)

3. (Cancelled.)

4. (Currently Amended) A method as in claim 1, further comprising the step of storing the standby archive ~~data file~~transaction redo log on the standby site.

5. (Currently Amended) A method as in claim 1, further comprising the step of storing transactions in the standby active ~~data file~~transaction redo log as they are trapped at the primary site.

A1
B1

6. (Currently Amended) A method as in claim 1, further comprising the step of entering transactions from the archive data file transaction redo log into the standby transaction processing system.

7. A method for creating archive redo logs for a standby database at a remote location, comprising the steps of:

- (a) trapping active log writes using a remote mirroring device that is capable of mirroring the log writes to a standby site, to create an online redo log at the standby site;
- (b) computing a difference between an online redo log and archive redo log to create a delta image at a primary database;
- (c) transmitting the delta image to a standby archive log replicator contained in the standby database; and
- (d) combining the delta image with the online redo log to create a standby archive redo log on the standby database.

8. A method as in claim 7, further comprising the step of applying the standby archive redo log by creating links to the standby archive redo log.

9. A method as in claim 7, further comprising the step of transmitting the delta image to a standby archive log replicator using a first remote mirroring device in the primary database and a second remote mirroring device in the standby database.

At
B1

10. A method as in claim 9, further comprising the step of using replication handles to track the progress of mirror writes on the standby database as compared to the primary database.

11. A method as in claim 10, further comprising the step of checking the primary database queries using the archive log replicator to determine whether archiving is complete.

12. A method as in claim 7, further comprising the step of shipping an entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously after archiving becomes slow enough that there is a risk of primary database blocking.

13. A method as in claim 12, further comprising the step of shipping an entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously when a spool buffer in a remote mirroring system has been filled.

14. A method for creating archive redo logs from a primary database, for a standby database at a remote location, comprising the steps of:

(a) trapping file system transactions to create an online redo log at the standby database;

(b) computing a difference between an online redo log and archive redo log in the primary database to find the delta image of the archive redo log;

(c) transmitting the delta image to a standby database; and

(d) combining the delta image with the online redo log in the standby database to create standby archive redo log on the standby database.

A
B1

15. A method as in claim 14, further comprising the step of applying the standby archive redo log by creating a link between the standby database and the standby archive redo log.

16. A method as in claim 14, further comprising the step of transmitting the delta image to a standby archive log replicator using a first remote mirroring device in the primary database and a second remote mirroring device in the standby database.

17. A method as in claim 16, further comprising the step of using replication handles to track the progress of mirror writes on the standby database as compared to the primary database.

18. A method as in claim 14, further comprising the step of checking the primary database with database queries using the archive log replicator to determine whether archiving is complete.

19. A method as in claim 14, further comprising the step of shipping an entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously after archiving becomes slow enough that there is a risk of the primary database blocking.

20. A method as in claim 14, further comprising the step of shipping the entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously when a spool buffer in a remote mirroring system has been filled.

AT
B

21. A system for creating a standby data processing system located at a standby site, which is remotely located from a primary site, comprising the steps of:

means for computing a difference between an active data file and an archive data file to produce a delta image at the primary site;

means for transmitting the active data file and the delta image, separately in time sequence, from the primary site to the standby site; and

means for combining the delta image with the active data file at the standby site, to produce a standby archive data file.

22. An article of manufacture comprising:

a computer usage medium having computer readable program code means embodied therein for creating archive redo logs for a standby database at a remote location, the computer readable program code means in said article of manufacture comprising:

computer readable program code means for trapping active log writes using a remote mirroring device that is capable of mirroring the log writes to a standby site, to create an online redo log at the standby site;

computer readable program code means for computing a difference between an online redo log and archive redo log to create a delta image at a primary database;

computer readable program code means for transmitting the delta image to a standby archive log replicator contained in the standby database; and

computer readable program code means for combining the delta image with the online redo log to create a standby archive redo log on the standby database.

Remarks

Applicant expresses appreciation to the Examiner for the consideration of the subject patent application.

The Office Action rejected claims 1-22 under 35 U.S.C. 102(e), as being anticipated by Tamer (U.S. No. 6,035,412). FIG. 1 of Tamer illustrates that Tamer is only interested in the interaction of the local mirroring between the two data storage systems 2 and 4. These two data storage systems in Tamer loosely correspond to 50 and 52 in FIG. 3 of the present invention.

The prior art does not teach or suggest transmitting a delta image for an archive transaction redo log to a standby site as claimed in the present invention. Specifically, claim 1 includes the patentably distinct language of “transmitting the active transaction redo log and the delta image, separately in time sequence, from the primary site to the standby site.” Transferring a delta image over a WAN to a standby cluster reduces data transfer on the WAN because the archive transaction redo log does not need to be shipped.

The Office Action stated that Tamer teaches a difference between an active data file and an archive data file to produce a delta image. What Tamer actually teaches is the use of a bitmap to identify which portions of the backup database are out of sync with the primary database (Col. 7, lines 40-64). Tamer does not teach computing a difference between an active transaction redo log and an archive transaction redo log to create a delta file. The delta image that is created by computing the difference between the transaction redo log and the archive transaction log is not a bit map of “dirty tracks”. This is discussed in the disclosure of the present invention on page 10, lines 19-23 and page 11 lines 1-2.

The delta image includes the archive header and additional important differences between an active transaction redo log and an archive transaction redo log. More specifically, the delta image of the present invention is not used to copy parts of the active transaction redo log into the archive transaction redo log like the bit maps of Tamer. Essentially, the redo log portion of the active transaction redo log and the archive transaction redo log will be the same. The active transaction redo log is shipped over the WAN with the delta image, which includes the file header information. The delta image is then overlaid on

the active transaction redo log to create the archive transaction redo log. The delta file of the present invention does not track any data transfer between the active transaction redo log and the archive transaction redo log. Rather, it effectuates the speedy creation of the archive transaction redo log at the standby site.

In order to differentiate the present invention from the prior art, claim 1 has been amended to claim “computing a difference between an active transaction redo log and an archive transaction redo log to produce a delta image at the primary site”. This subject matter has been incorporated from claim 2 and 3. Please cancel claims 2 and 3.

Both the present invention and Tamer include bit maps or what are known as “dirty track maps” but in Tamer these bit maps are applied to the local mirroring as illustrated in FIG. 1 of Tamer where mirroring occurs between the first local data storage system 2 and the second local data storage system 4. (Column 7, lines 41-67) In one embodiment, the present invention uses bit maps in its remote mirroring system 66, 68 but the bit maps are separate and distinct from the delta image. As discussed above, Tamer does not teach or suggest enabling the creation of an archive transaction redo log on a standby site by combining a delta image with an online transaction redo log as claimed.

With regard to claim 5, Tamer does not teach “storing transactions in the standby active transaction redo log as they are trapped at the primary site.” Specifically, Tamer does not teach or suggest any operations with respect to a standby site. Particularly, Tamer describes two local data storage systems that are connected by a high speed link 6. (Col. 7 lines 33-36.) In contrast, the present invention focuses on reducing the amount of data transfer to a standby cluster 53 over a lower bandwidth network as illustrated in FIG. 3 by the WAN 73. The master and slave, described in column 4, lines 32-49 of Tamer, refer only to a data storage system located at a primary site and not at a standby site. Claim 6 includes further patentably distinct subject matter when considered in light of claim 1 from which claim 6 depends.

Claims 7-22 were also rejected by the Office Action under the same arguments applied to claims 1-6. The method claims of present invention also include the steps of combining the online transaction log 80 with the delta image to form the archive transaction

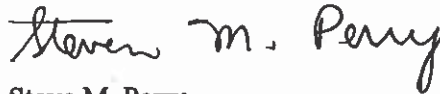
log(s) 76, 78 at the standby site, and this is not taught or suggested by the prior art. The same arguments that have been presented by the Applicant also apply to these claims 7-22 and claims 1-22 should be allowed.

If any impediment to the allowance of these claims remains after consideration of this Amendment, the Examiner is invited to call Steve Perry at (801) 566-6633 so that any necessary changes to the claims can be considered.

The Commissioner is hereby authorized to charge any additional fee or to credit any overpayment in connection with this Amendment to Deposit Account No. 08-2025.

DATED this 22nd day of April, 2003.

Respectfully submitted,



Steve M. Perry
Attorney for Applicant
Registration No. 45,357

THORPE, NORTH & WESTERN, L.L.P.
P.O. Box 1219
Sandy, Utah 84091-1219
Telephone (801) 566-6633



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/774,839	01/31/2001	Shivaji Ganesh	10006924-1	6266

7590 08/05/2003
HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, CO 80527-2400

EXAMINER

KINDRED, ALFORD W

ART UNIT PAPER NUMBER

2172

DATE MAILED: 08/05/2003

6

Please find below and/or attached an Office communication concerning this application or proceeding.

DETAILED ACTION

1. This action is responsive to communication: Application A, filed on 04/28/03.
This action is made final.

Claim Rejections - 35 USC § 103

2. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

3. Claims 1 and 4 and 6-22 are rejected under 35 U.S.C. 103(a) as being unpatentable over Tamer et al., US# 6,035,412, in view of Bohannon et al., US# 2002/0091718 A1.

As per claims 1 and 4, Tamer et al. teaches "a difference between an active transaction redo log and an archive transaction redo log to produce a delta image . . ." (see abstract) "transmitting the active transaction redo log and the delta image . . ." (see fig. 1A—sheet 1 of 15, whereas Tamer's "Bit Maps" is equivalent to applicant's "image" data) "combining the delta image with the active . . ." (see col. 7, lines 40-60). Tamer et al. does not explicitly teach "combining the delta image with the active transaction redo log at the standby site, to produce a standby archive . . .". Bohannon et al. teaches "combining the delta image with the active transaction redo log at the standby site, to produce a standby archive . . ." (page 3, paragraph [0030] and page 8 [0099]. It would have been obvious at the time of the invention for one of ordinary skill in the art to have

combined the teachings of Tamer and Bohannon, because using the steps of "combining the delta image with the active transaction redo log at the standby site, to produce a standby archive . . ." would have given those skilled in the art the tools to combined the image data with transactional active and archived transactional logs. This give users the advantage of processing transactional data more efficiently.

As per claim5, Tamer et al. teaches "storing transactions in the standby active . . ." (see col. 4, lines 32-49).

As per claim 6, Tamer et al. teaches "entering transactions from the archive data file into the standby transaction . . ." (see col. 2, lines 50-62).

As per claims 7-13 and 14-20, these claims are rejected on grounds corresponding to the arguments given above for rejected claims 1 and 4-6 and are similarly rejected including the following:

--Tamer et al. Teaches "mirroring device . . ." (see col. 3, lines 24-40).

As per claims 21-22, these claims are rejected on grounds corresponding to the arguments given above for rejected claims 1-2 and are similarly rejected.

Response to Arguments

4. Applicant's arguments with respect to claims 1 and 4-6 have been considered but are moot in view of the new ground(s) of rejection.

5. Applicant's arguments filed 4/28/03, with regards to claims 7-22 have been fully considered but they are not persuasive. As per applicant's arguments regarding "the prior art does not teach or suggest transmitting a delta image for an archive transaction

Application/Control Number: 09/774,839
Art Unit: 2172

Page 4

redo log to a standby site . . .” examiner maintains that Tamer's teachings of synching portions relating to a backup database in relations with a primary database teaches applicant's claim language of computing the difference between transaction logs.

Conclusion

6. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire **THREE MONTHS** from the mailing date of this action. In the event a first reply is filed within **TWO MONTHS** of the mailing date of this final action and the advisory action is not mailed until after the end of the **THREE-MONTH** shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than **SIX MONTHS** from the date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Alford W. Kindred whose telephone number is 703-305-3802. The examiner can normally be reached on Mon-Friday, 8:00am-5:00pm.

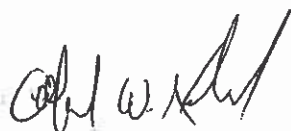
If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kim Vu can be reached on 703-305-4393. The fax phone numbers for the

Application/Control Number: 09/774,839
Art Unit: 2172

Page 6

organization where this application or proceeding is assigned are 703-746-7239 for regular communications and 703-746-7238 for After Final communications.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is 703-305-3900.



Alford W. Kindred
Patent Examiner
Tech Ctr. 2100
July 31, 2003

Notice of References Cited	SWEET ET AL.	
	Examiner Alford W. Kindred	Art Unit 2172
Page 1 of 1		

U.S. PATENT DOCUMENTS					
*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-2002/0091718 A1	07-2002	BOHANNON et al.	707/202
	B	US-			
	C	US-			
	D	US-			
	E	US-			
	F	US-			
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS						
*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS		
*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	
	V	
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.



US 20020091718A1

(19) **United States**

(12) **Patent Application Publication**
BOHANNON et al.

(10) **Pub. No.: US 2002/0091718 A1**
(43) **Pub. Date: Jul. 11, 2002**

(54) **METHOD AND APPARATUS FOR
DETECTING AND RECOVERING FROM
DATA CORRUPTION OF DATABASE VIA
READ LOGGING**

Related U.S. Application Data

(63) Non-provisional of provisional application No. 60/099,265, filed on Sep. 4, 1998. Non-provisional of provisional application No. 60/099,271, filed on Sep. 4, 1998.

(76) **Inventors:** PHILIP L. BOHANNON, MORRIS,
NJ (US); RAJEEV RASTOGI,
UNION, NJ (US); SRINIVASAN
SESHADRI, BASKING RIDGE, NJ
(US); ABRAHAM SILBERSCHATZ,
UNION, NJ (US);
SUNDARARAJARAO SUDARSHAN,
POWAI, BOMBAY (IN)

Publication Classification

(51) **Int. Cl.⁷** G06F 12/00
(52) **U.S. Cl.** 707/202

(57) **ABSTRACT**

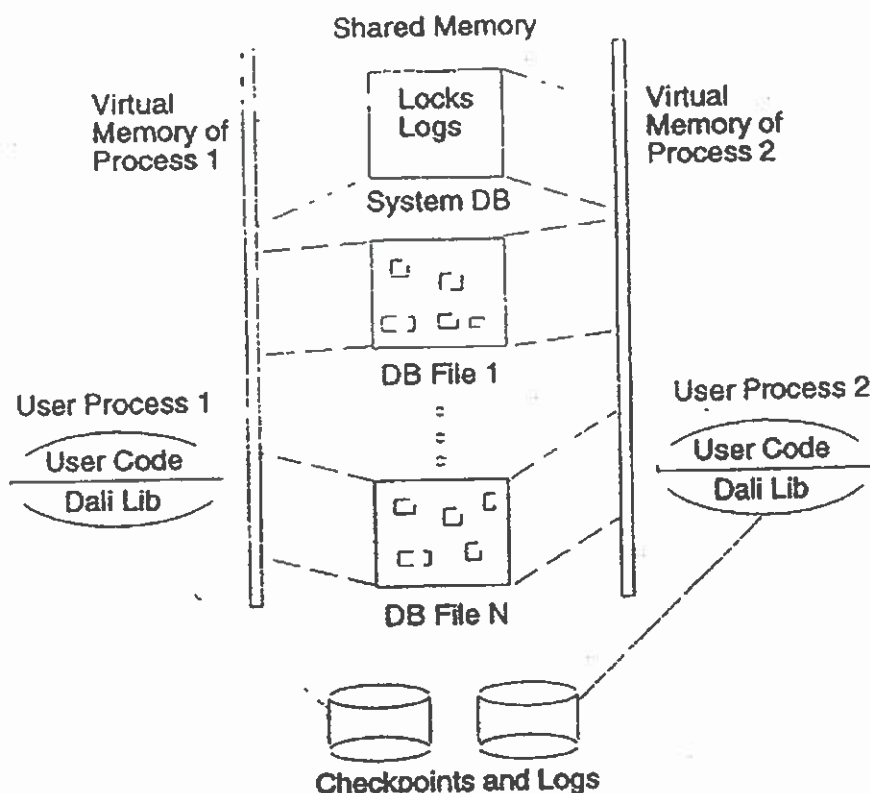
A method of detecting and recovering from data corruption of a database is characterized by the step of logging information about reads of a database in memory to detect errors in data of the database, wherein said errors in data of said database arise from one of bad writes of data to the database, of erroneous input of data to the database by users and of logical errors in code of a transaction. The read logging method may be implemented in a plurality of database recovery models including a cache-recovery model, a prior state model a redo-transaction model and a delete transaction model. In the delete transaction model, it is assumed that logical information is not available to allow a redo of transactions after a possible error and the effects of transactions that read corrupted data are deleted from history and any data written by a transaction reading Ararat data is treated as corrupted.

Correspondence Address:
THOMAS H. JACKSON
BANNER AND WITCOFF LTD
11TH FLOOR
1001 G STREET NW
WASHINGTON, DC 200014597

(*) **Notice:** This is a publication of a continued prosecution application (CPA) filed under 37 CFR 1.53(d).

(21) **Appl. No.: 09/207,927**

(22) **Filed: Dec. 9, 1998**



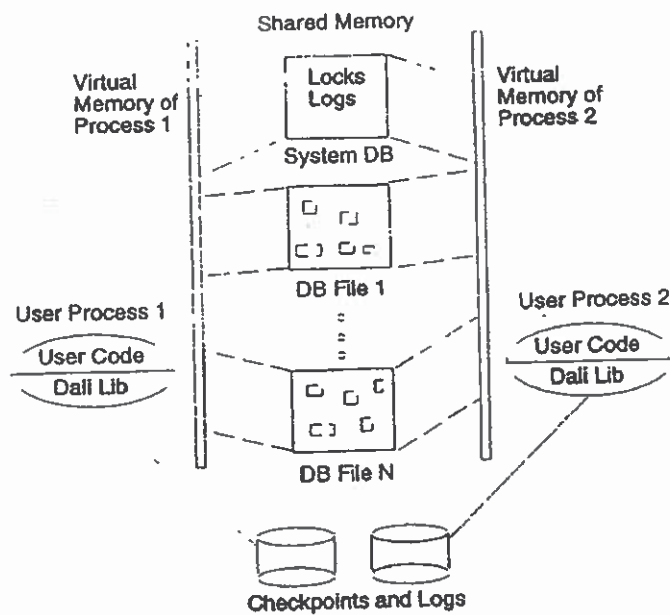


Figure 1

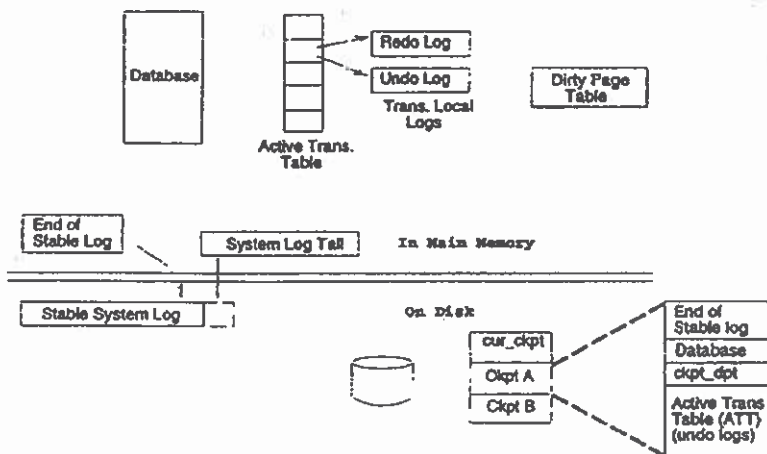


FIGURE 2

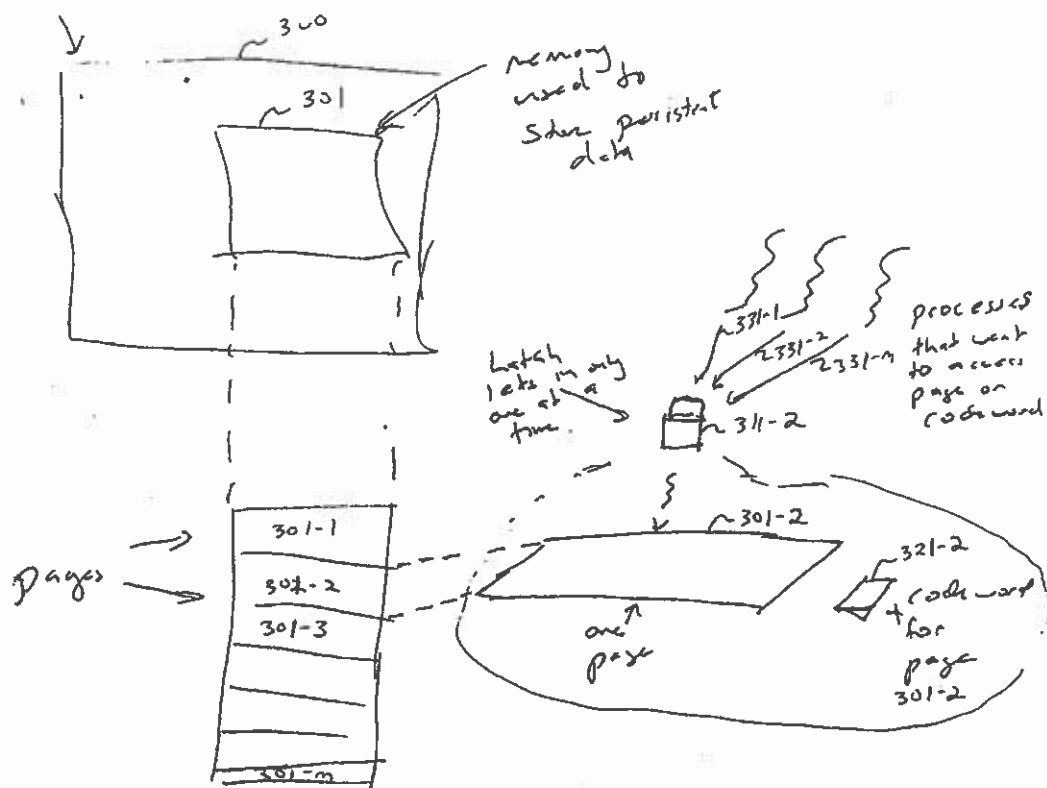


Fig 3, Read Latching for Simple prechecking scheme

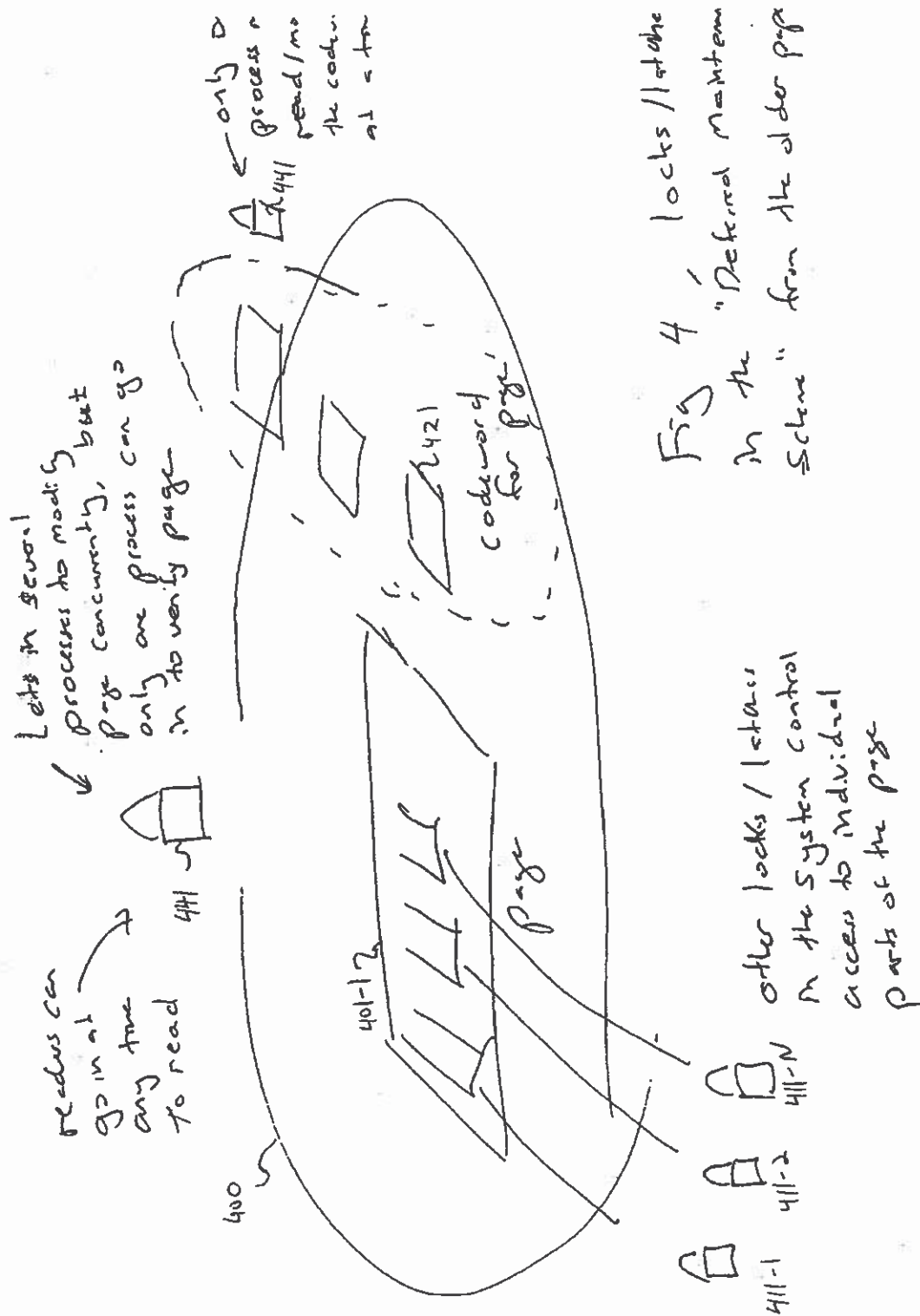


Fig. 4, locks / latches in the "Deferred Maintenance Scheme" from the older page

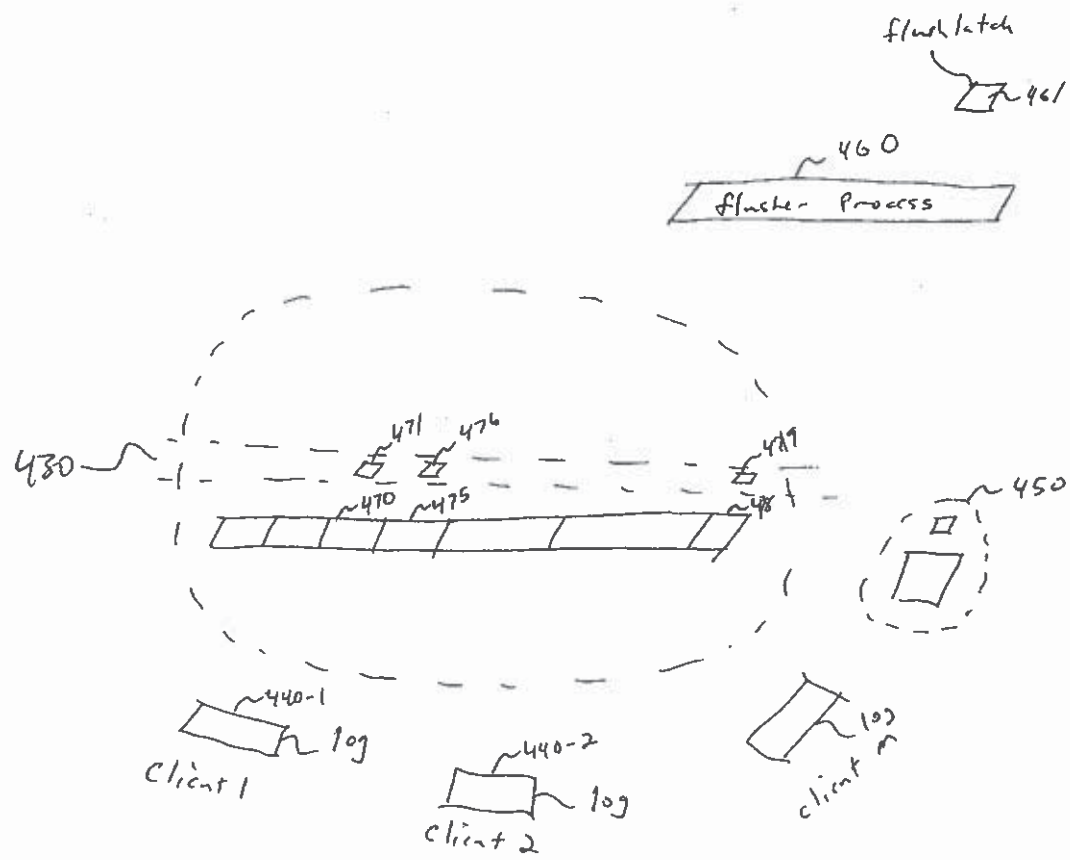
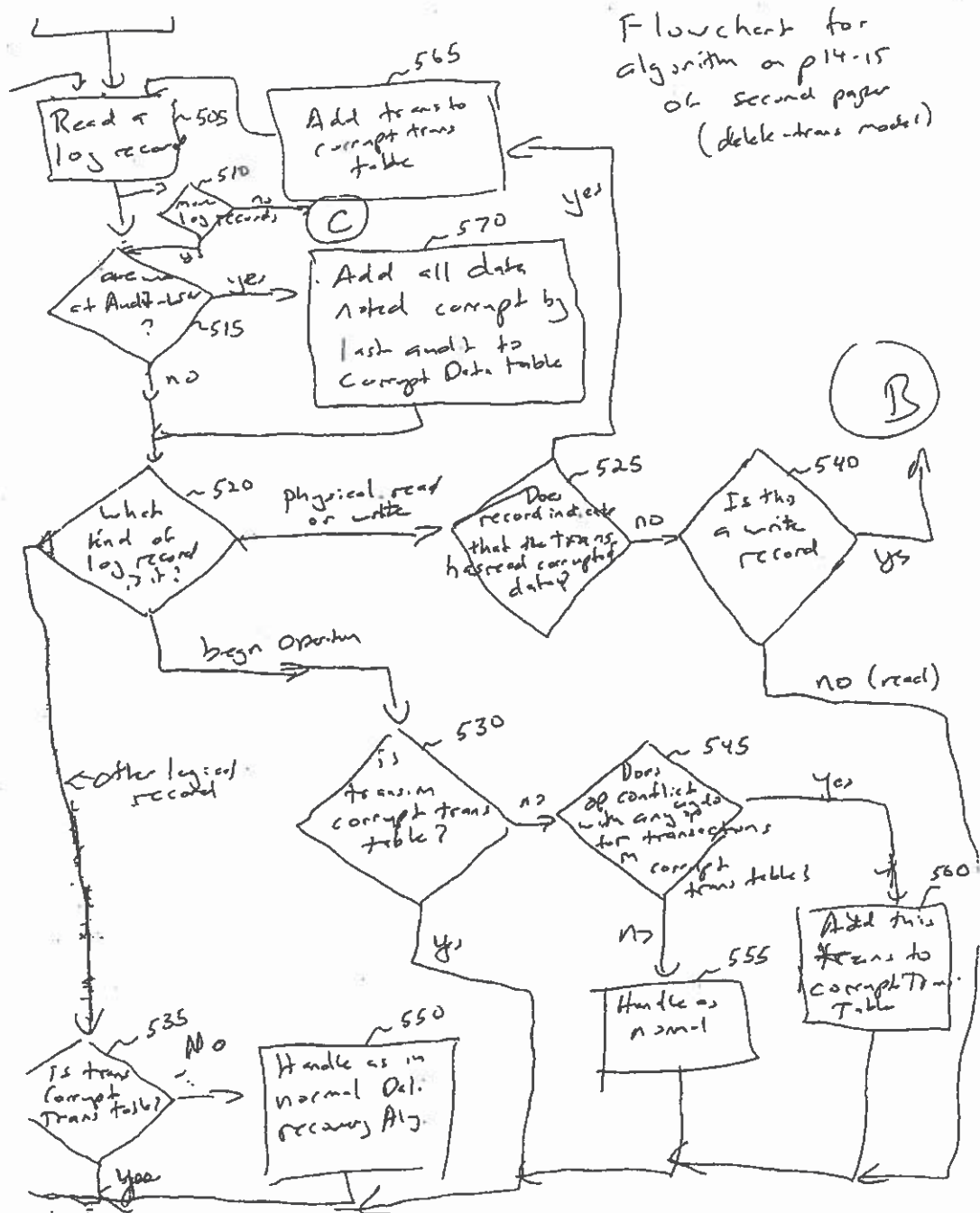
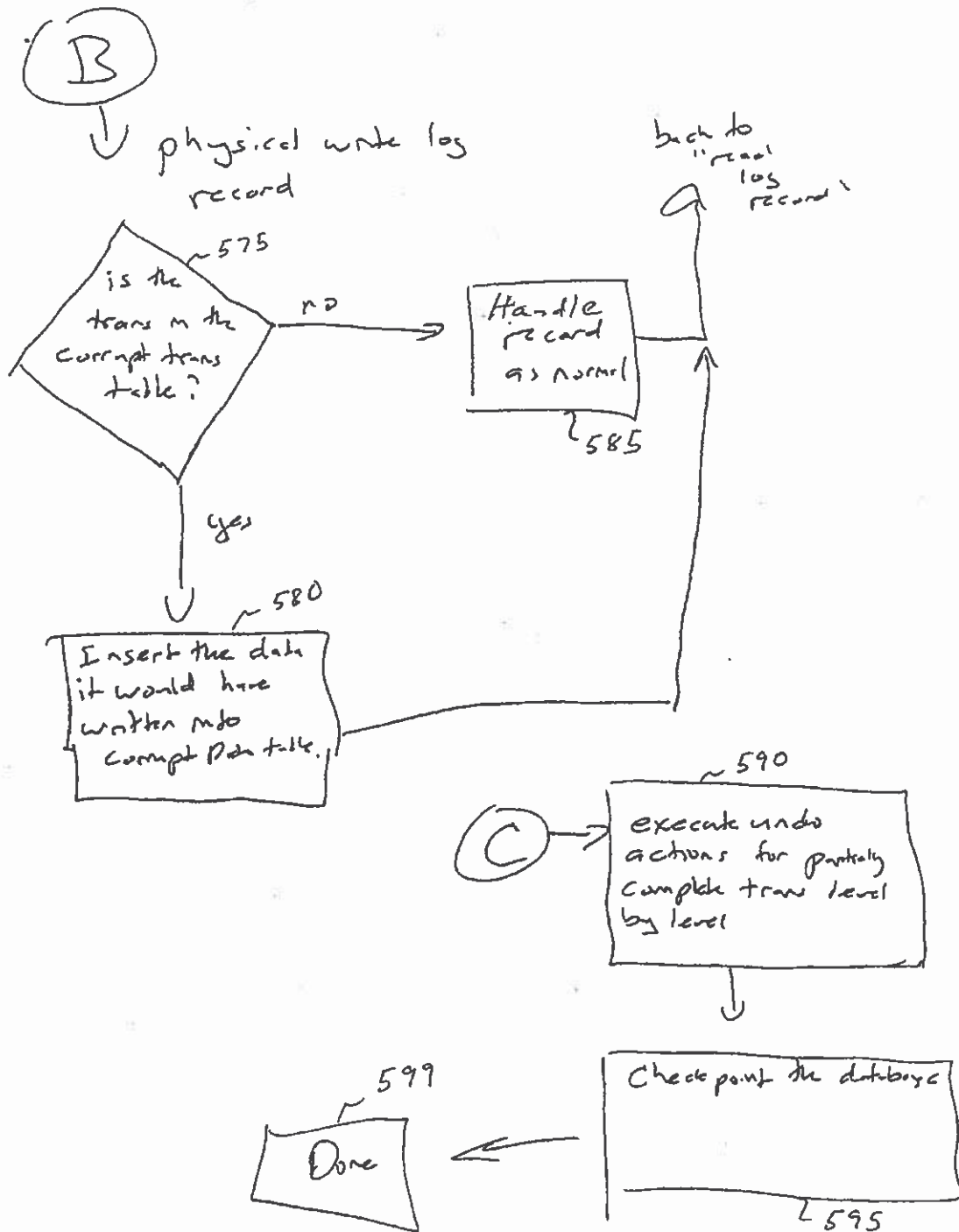


FIG 4B





METHOD AND APPARATUS FOR DETECTING AND RECOVERING FROM DATA CORRUPTION OF DATABASE VIA READ LOGGING

[0001] This application claims priority and is related by subject matter to U.S. application Ser. No. 08/766,096, entitled "System and Method for Restoring a Distributed Checkpointed Database," of Bohannon et al., to U.S. application Ser. No. 08/767,048, entitled "System and Method for restoring a Multiple Checkpointed Database in View of Loss of Volatile Memory" of Bohannon et al., both applications being filed Dec. 16, 1996 and to U.S. Provisional Patent Application Serial No.'s 60/099,265 and 60/099,271, filed Sep. 4, 1998 of Bohannon et al.

BACKGROUND OF THE INVENTION

[0002] 1. Technical Field

[0003] The present invention relates to the field of database management systems generally and, more particularly, to method and apparatus for detecting and recovering from data corruption of a database by codeworded regions of the database and by logging information about reads of the database.

[0004] 2. Description of the Related Arts

[0005] A database is a collection of data organized usefully and fundamental to many software applications. The database is associated with a database manager and together with its software application comprises a database management system (DBMS). In recent years, extensible database systems such as Illustra (now part of the Informix Universal Server) have been developed which allow the integration of application code with database system code. In these systems, the application code has direct access to the buffer cache and other internal structures of the DBMS. Similarly, application programs in many object oriented database (OODB) systems have direct access to an object cache in their address space. This OODB architecture was developed to minimize the cost of accesses to data, for example, to support the needs of Computer Aided Design (CAD) systems. Also, several recently developed storage management systems provide memory resident or memory mapped architectures. For example, the Dali main-memory storage manager described in Bohannon et al., "The Architecture of the Dali Main-Memory Storage Manager," *Journal of Multimedia Tools and Applications*, 4(2), pp. 115-151 (1997) is designed to provide applications with fast, direct access to data by keeping the entire database in volatile main memory. In all these systems, direct access to data (either in the database buffer cache or in a memory-mapped portion of the database) by application programs is critical to providing fast response times. The alternative to memory mapping is to access data via a server process, but this presents an unacceptable solution due to the high cost of inter-process communication. Application code is typically less trustworthy than database system code, and there is therefore a significant risk that "wild writes" and other programming errors can affect persistent data in systems that allow applications to access such data directly. Since the systems described above are increasingly popular, the risk of wild writes and associated physical corruption is growing. Additionally, there is a risk of damage due to software faults in the DBMS itself. It is therefore important to develop techniques that can mitigate the risk of corruption.

[0006] In our parent U.S. patent application Ser. No. 08/766,096, filed Dec. 16, 1996 and entitled "System and Method for Restoring a Distributed Checkpointed Database," we describe the application of multiple checkpoints and the maintenance of a stable log record stored on a server for tracking operations to be made to the multiple checkpoints in a distributed environment. A companion parent application, U.S. patent application Ser. No. 08/767,048, entitled "System and Method for Restoring a Multiple Checkpointed Database in View of Loss of Volatile Memory" filed the same day describes recovery processes at multiple levels of a DBMS in the event of loss of volatile memory. The '048 and the '096 applications should be deemed to be incorporated by reference herein as to their entire contents. Both of these applications relate to the preservation and restoration of a database (or distributed database), for example, stored in main volatile memory of a data processor.

[0007] The problem of detecting and recovering from corruption of data in a database system still remains to be solved in a pragmatic manner without adding considerable overhead to the DBMS. Data corruption may be physical or logical and it may be direct or indirect. Data is "directly" corrupted in a physical corruption sense by "unintended" updates, such as wild writes as explained above due to programming errors in the physical case, or arising from incorrectly coded updates or input errors (human errors) in the logical case. Once data is directly corrupted, it may be read by a process, which then issues writes based on the value read. Data written in this manner is indirectly corrupted, and the process involved is said to have carried the corruption. While this process may be a database maintenance process, we focus on transaction-carried corruption, a problem in which the carrying process is executing transactions.

[0008] Direct physical corruption can be mostly prevented with hardware memory protection, using the virtual memory support provided by most operating systems. One approach involves mapping the entire database in a protected mode, and selectively un-protecting and re-protecting pages as they are updated. However, this can be very expensive, for example, on standard UNIX systems. An alternative to the hardware approach would be programming language techniques such as type-safe languages or sandboxing. (Sandboxing is a technique whereby an assembly language programmer adds code immediately before a write to ensure that the instruction is not affecting protected space.) However, type-safe languages have yet to be proven in high-performance situations, and sandboxing may perform poorly on certain architectures. Finally, communication across process domain boundaries to a database server process provides protection, but such communication is orders of magnitude slower than access in the same process space, even with highly tuned implementations. The concern over physical corruption is further motivated by the increasing number of systems in which application code has direct access to system buffers, including extensible systems, object databases, and memory-mapped or in-memory architectures. Finally, some work has raised concern over damage to data due to faults in the DBMS itself.

[0009] Integrity constraints are widely studied and prevent certain cases of logical corruption in which rules about the data would be violated. However, it is an object of the

present invention to deal with those cases in which integrity constraints and other input validation techniques fail, and whether due to programming error or invalid input, unintended updates are made to the database. We consider such cases inherently impossible to prevent, and instead assume that the problem is detected later, usually when a database user notices incorrect output (on a bank statement, for example).

[0010] In the field of accounting systems and audits, it is known from L. A. Bjork, Jr., "Generalized Audit Trail Requirements and Concepts for Data Base Applications", *IBM Systems Journal*, No. 3, 1975, pp. 229-245, how to create and maintain an audit trail—a history of activities by transaction, posted because of operations on specific data. Bjork describes that a time dimension can be added to a stored record such that supplemental information in the form of a descriptor and time frame are maintained. For example, the time frame when the information was created and stored, and each version of a data field, is maintained along with the action. He refers to "create" (creation of data), "reference" (when reference is made to x at time t), and update (when created data is updated) as descriptors all having time dimension. A further descriptor is "refer to prior generation" (when data now updated is referred to by a prior generation." Also, C. T. Davies, Jr. in his article "Data Processing Spheres of Contro" appearing in the *IBM Systems Journal*, Vol. 17, No. 2, 1978, pp. 179-198 describes "in-process recovery" as the control of recording and subsequent use of data required to return to a previous point in a process, and that the process that created or last modified data elements be determined from a journal. System recovery is obtained via establishing checkpoints that represent an early state in a data base. Once a search backward is conducted to find an error, a checkpoint behind the error is obtained from which to rebuild. These accounting processes, for example, typified by the recovery from a payroll error, are not examined by Bjork or Davies for the generic case of database recovery, nor are they automated. Bjork and Davies provide no suggestions for real-time implementation, for example, of a read-logging recovery system such as would be required in a communications system and associated record-keeping environment.

[0011] Thus, there appears a genuine need in the art of database management systems to provide an improved method and apparatus for detecting and recovering from corruption of a database via read logging.

SUMMARY OF THE INVENTION

[0012] According to the present invention, it is a principle to apply several new techniques for the prevention or detection of corruption. The new techniques may be suitable for application in a real-time environment such as in a telecommunications system.

[0013] For detecting indirect logical or physical corruption, it is a feature of the present invention to log information about reads (Read Logging). Interestingly, any negative impact of Read Logging is limited, as the actual values read are not logged according to one embodiment of the present invention, just the identity of the item read and optionally a checksum of the value. Moreover, it is an extension of the present invention to store codewords for each read of the read log records.

[0014] When corruption is detected rather than prevented, techniques for corruption recovery are employed to restore the database to an uncorrupted state. As will be further described herein, once codewording of data and read logging is performed, models and algorithms are presented for recovery from transaction-carried indirect corruption. In these models, the read log records are preferably combined with known write log records and operated on more efficiently as a combined log to detect and recover from transaction based corruption (although in other embodiments of the present invention, the read log and write log records may be separately maintained.) One model, the redo-transaction model, uses logical descriptions of transactions to repair the database state. A second model, the delete-transaction model, focuses on removing the effects of corruption from the database image. The algorithms presented herein can be applied to recovery from logical or physical corruption. In addition, tracing techniques are presented which aid in determining the scope of logical corruption.

[0015] To ascertain the performance of our algorithms for detecting and recovering from physical corruption, we have studied the impact of these schemes on a TPC-B style workload implemented in the Dali main-memory storage manager. Our goal was to evaluate the relative impact on normal processing of schemes that can be easily ported across a variety of architectures and operating systems. In addition to our schemes, we study a hardware-based protection technique. For detection of direct corruption, the overheads imposed cause throughput of update transactions to be decreased by 8%. Prevention of transaction-carried corruption with Read Prechecking costs between 12% and 72%, but requires a significant space overhead to achieve the better performance numbers. Detection of transaction-carried corruption with Read Logging costs between 17% and 22%. Our study indicates that the corruption prevention algorithms of Sullivan et al., "Using Write Protected Data Structures to Improve Fault Tolerance in Highly Available DBMS" in *Proceedings of the International Conference on Very Large Databases*, pp 171-179, 1991, when using standard OS support for memory protection, decrease throughput by about 38%. Thus, the codeword and read logging based detection and prevention schemes of the present invention perform significantly better than the hardware-based protection.

[0016] With the present invention, it is possible to identify a subset of the later transactions that were (directly or indirectly) affected by the error, and to selectively roll them back and redo them manually (or even automatically in some cases). Also, the techniques of the present invention are language and instruction-set independent.

[0017] Thus, a method of detecting and recovering from data corruption of a database according to the present invention is characterized by the step of logging information about reads of a database to detect and recover from physical corruption of the data in the database, wherein the physical corruption arises from bad writes of data to the database or arises indirectly from the bad writes. In a delete transaction model, the corruption recovery comprises first and second phases, a first redo phase followed by an undo phase. In another embodiment, a method of detecting and recovering from database corruption comprises the steps of logging information about reads of a database to detect and recover

from logical corruption, maintaining a logical redo log and storing user inputs for a transaction in the logical redo log. Alternatively, in the logical corruption recovery method, the method comprises the step of logging a checksum of a logical state found.

[0018] In our co-pending, concurrently filed patent application, we describe a Read Prechecking scheme that associates one word codewords with each region of data, and prevents transaction-carried corruption by verifying that the codeword matches the data each time it is read. A Data Codeword scheme, a less expensive variant of Read Prechecking, allows detection of direct physical corruption by asynchronously auditing the codewords. This scheme is also referred to herein and in our co-pending application as deferred codeword maintenance and involves performing codeword updates during a process called "log flushing" at the same time as data is flushed to disc from main memory. These schemes are disclosed and claimed in concurrently filed, copending U.S. patent application Ser. No. (Attorney Docket Number Bohannon et al. 8-25-3-38-11) entitled "Method and Apparatus for Detecting and Recovering from Data Corruption via Read Prechecking and Deferred Maintenance of Codewords," of the same inventors.

[0019] These and other features of the present invention will be best understood from considering the drawings and the following detailed description of the preferred embodiments.

BRIEF DESCRIPTION OF THE DRAWINGS

[0020] FIG. 1 is a functional schematic drawing of the overall architecture of the Dali main memory storage manager useful for explaining an implementation of the present invention.

[0021] FIG. 2 is a functional block diagram providing an overview of database recovery structures which may be existent, for example, within an architecture of FIG. 1.

[0022] FIG. 3 shows a database memory comprising a plurality of pages and codewords for each page with a latch for processes wanting access to a page or a codeword for that page useful in describing a simple read prechecking scheme.

[0023] FIG. 4A also shows a database memory organized with codewords for page regions having a page portion protection latch permitting only one process to read or modify a codeword at a time useful in describing deferred codeword maintenance and FIG. 4B is useful in describing how deferred maintenance operates at log flush and associated recovery and audit processes.

[0024] FIGS. 5A and 5B together comprise a flowchart of a recovery algorithm for the delete transaction model of the present invention comprising a redo phase followed by an undo phase before checkpointing the database.

[0025] FIG. 6 comprises an alternative algorithmic approach to recovery for the redo-transaction model.

DETAILED DESCRIPTION OF THE PRESENT INVENTION

[0026] Introduction

[0027] Referring briefly to FIG. 1, there is shown an overall architecture of the Dali storage manager on which the present invention has been implemented. The Dali system should be considered only an exemplary implementation of the present invention and the present detection and recovery algorithms may be applied in other architectures as well. The database in a Dali system consists of one or more database files, along with a special system database file. User data itself is stored in database files while all data related to database support, such as log and lock data, is stored in the system database file. This enables storage allocation routines to be uniformly used for persistent user data as well as non-persistent system data like locks and logs. The Dali system database file also persistently stores information about the database files in the system.

[0028] As shown, database files opened by a process are directly mapped into the address space of that process. In Dali, either memory-mapped files or shared-memory segments can be used to provide this mapping. Different processes may map different sets of database files, and may map the same database file to different locations in their address space.

[0029] This feature precludes using virtual memory addresses as physical pointers to data (in database files), but provides two benefits. First, a database file may be easily resized. Second, the total active database space on the system may exceed the addressing space of a single process. This is useful on machines with 32-bit addressing in which physical memory can significantly exceed the amount of memory addressable by a single process. In a 64-bit machine, both of these considerations may be mitigated, so we also consider physical addressing. For example, if a single database file can be limited to the order of 64 gigabytes, then each process can still map close to a billion database files (which can be expected to far exceed the total database space).

[0030] There is a single active transaction table (ATT), stored in the system database, that stores redo and undo logs for each active transaction. A dirty page table (dpt) is maintained for the database (also in the system database) which records the pages that have been updated since the last checkpoint. The ATT (with undo logs) and the dirty page table are also stored with each checkpoint. The dirty page table (dpt) in a checkpoint is referred to as ckpt_dpt in the figure.

[0031] Models and Overview

[0032] In this section we describe our models of corruption and recovery and the database model in which our algorithms are described.

[0033] Models of Corruption

[0034] As described above, data corruption may be either direct or indirect, logical or physical. Direct physical corruption is defined to be an update to the in-memory image of database data which did not happen through prescribed methods. For example, in a standard disk-based system, an update to an internal buffer without a latch and without calls to logging routines (usually made via a FixBuffer or similar

function call) would constitute direct physical corruption. For applications with direct access to data, such corruption could result due to application errors (e.g., stray pointers in the application code).

[0035] Examples of direct logical corruption include any update made in error, for example, an accidental deletion of a customer, updates to account balances made by a program which computed interest incorrectly, or simply entering an incorrect amount for sales commission which is then used in payroll computations. Once direct corruption has occurred, corrupted data may be read by a subsequent transaction. If that transaction, considered a corrupt transaction, then writes other data in the database, that data is also assumed to be corrupt. Note that the newly "corrupted" data may then be read by other transactions, and the corruption propagated further. We call such corruption indirect corruption.

[0036] As with corruption of user data, corruption of control structures may be handled with hardware memory protection, or with codeword based techniques.

[0037] Models of Recovery

[0038] We define four models of corruption recovery: a cache recovery model, a prior-state model, and a redo transaction model and a delete-transaction model. The first model deals only with direct physical corruption. The other models deal with logical and indirect physical corruption.

[0039] In the cache-recovery model, corruption is removed from cache pages, assuming that indirect corruption has not occurred, and because of that, corrupt data values are not reflected in any log records.

[0040] In the prior-state model, the goal is to return the database to a transaction consistent state prior to the first possible occurrence of corruption. Most commercial systems support this model. Clearly, this recovery model leaves much to be desired, as it may well be impractical to request that users re-submit any work done following the introduction of the corruption.

[0041] In an extension of the prior-state model, once the direct error has been identified and corrected, transactions affected by the error must be logically re-run in history; that is, the transactions must be re-run in the same serialization order as the original set of transactions. We call this extension of the prior-state model the redo-transaction model of corruption recovery.

[0042] In our final model, the delete-transaction model, we assume that logical information is not available to allow a redo, and corruption is dealt with by deleting the effects of transactions from the database image. Any transaction that read corrupted data must be deleted from history, and any data that such a transaction wrote after reading corrupt data is treated as being corrupted by the transaction.

[0043] To implement a recovery algorithm for the delete transaction model, it must be clearly understood what it means to "delete a transaction from history". One possible interpretation would be to allow any serializable execution of the remaining, undeleted, transactions. However, this definition is not acceptable, since the values read by other transactions, and thus the values exposed to the outside world, might change in the modified history.

[0044] To define correctness in the delete transaction model, we consider two transaction execution histories, the

original history, H_o , and the delete history, H_d , in which all reads and writes of certain transactions no longer appear. These histories include the values read or written by each operation, and for a given operation, the value read or written in the delete history is the same as in the original history. A delete history is conflict-consistent with the original history if any read in H_d is preceded by the same write which preceded it in H_o . Similarly, H_d is view-consistent with H_o if each read in H_d returns the value returned to it in H_o . Note that the notions of conflict- or view-consistency are distinct from the standard notions of conflict- or view-equivalence. A correct recovery algorithm in the delete-transaction model recovers the database according to a delete history which is conflict- or view- consistent with the original history. Note that it follows from this definition that in a conflict-consistent delete history, the final state of any data item written by a transaction in the delete set will have the value it had before being written by the first deleted transaction.

[0045] Levels of Protection

[0046] Having covered the models of corruption and corruption recovery, we now consider different levels of protection. In general, it is possible to prevent or detect either direct or indirect physical corruption. It may further be possible to detect indirect logical corruption, once the fact that a certain transaction introduced the logical corruption has been determined by a human. The human then can initiate appropriate action. In each of these cases, if a form of corruption is detected but not prevented, a recovery mechanism for that corruption should exist.

[0047] Note that prevention of direct corruption is equivalent to prevention of all corruption, since other forms only propagate direct corruption. Preventing direct logical corruption is impossible, and preventing direct corruption using hardware may be expensive as discussed above. In the next best alternative, direct corruption is detected and transaction-carried corruption is prevented. Finally, direct corruption and transaction-carried corruption may both be detected. Detecting or preventing only transaction-carried corruption with no detection of direct corruption makes little sense, as the source of the corruption would remain in the database indefinitely. However, detection of only direct physical corruption may be used, if one feels that the corruption will usually be detected and recovery performed before the direct corruption leads to transaction-carried corruption. As unsatisfactory as it seems, such minimal protection is a great improvement over no protection at all. Corrupt data left in the database indefinitely is far more likely to cause problems than corruption removed after a few minutes or even hours.

[0048] Some of the schemes we describe below associate a codeword with a region of data known as a protection region. When data in the region is updated through the prescribed interface, the codeword is updated along with the data. By using word-wise parity, or a similar scheme, updates to the codeword can be made based only on the portion of the protection region which was actually updated—the remainder of the region need not be read.

[0049] Using Codewords To Detect Corruption

[0050] The basic (and well known) idea of codeword-based corruption detection is as follows. Each protection

region also stores a codeword for the region. Updates via the prescribed interface also update the codeword for the region (either immediately on update, or in a deferred fashion, as we shall see). If an update not via the prescribed interface occurs, the stored codeword will not be updated. Thus (with a very high probability assuming a good codeword scheme), if we compute the codeword for the region it will not match the stored code (we ensure that the deferred codeword updates have been performed before matching the codewords).

[0051] There are several issues in codeword based corruption detection, such as:

[0052] When and how to update the codewords (Immediately on update? Deferred to later?)

[0053] When and how to check for codeword mismatch (When flushing data to disk or archival copy?)

[0054] How to integrate codeword detection with concurrency control and recovery (ensuring concurrency levels stay high; ensuring recoverability from failures).

[0055] These issues are addressed later, in Section 5; in this section we introduce some basic requirements on the use of codewords for detecting corruption.

[0056] Error detecting codes, such as the Cyclic Redundancy Check (CRC) code are widely used, for example to verify integrity of sectors of disk systems. However, our algorithms have non-standard requirements on the codeword schemes that may be used:

[0057] 1. It should be possible to update a codeword incrementally when part of the protection region that it covers is updated. In particular, to avoid locking unrelated data and to provide a high degree of concurrency, we must be able to compute the new codeword using only the old value of the codeword, and the old and new values of the part of the region that is updated.

[0058] 2. Since the actual update of the codeword may be deferred, we require that the effect on the codeword of an update M to part of a region can be summarized as $\Delta(M)$. The update M consists of the old and new values of the updated part of the protection region, and hence $\Delta(M)$ must be a function of only the location and the old and new values of the updated area. In particular, $\Delta(M)$ should not depend on the old codeword. Since in one of our schemes $\Delta(M)$ must be stored in the redo log record, we require that the information in $\Delta(M)$ should be small, only about as big as the codeword itself. We also assume there is a codeword update operation $\oplus$, such that if the codeword for a region prior to an update M is C_{old} , the codeword after the update is

$$C_{new} = C_{old} \oplus \Delta(M)$$

[0059] We shall assume that there is a value, denoted by 0 , that results in no change to the codeword; that is, for all C , we have $C \oplus 0 = C$.

[0060] 3. There may be a sequence of updates to a region whose Δ s are applied out-of-order to the codeword. We assume that while an update is in progress it has exclusive access to the updated data;

such exclusive access can be ensured by means of region locks. Further, the codeword change $\Delta(M)$ is computed while the update M has exclusive access to the updated data. Given the above, we require that for any pair of updates M_1 and M_2

$$(C \oplus \Delta(M_1)) \oplus \Delta(M_2) = (C \oplus \Delta(M_2)) \oplus \Delta(M_1).$$

[0061] We often treat M as if it is composed of two updates, one from the initial value of the updated area U to a value of all zeros, and a second from all zeros to the value R it holds after M . We use $\Delta^-(U)$, which we call the undo value, to denote $\Delta(M')$ where M' is the update from U to all zeros, while $\Delta^+(R)$, which we call the redo value denotes $\Delta(M'')$ where M'' is the update from all zeros to R .

[0062] Thus, we have $C \oplus \Delta(M) = C \oplus \Delta^-(U) \oplus \Delta^+(R)$. We shall assume a function $\oplus$ that can be used to combine $\Delta^-(U)$ and $\Delta^+(R)$ such that $\Delta(M) = \Delta^-(U) \oplus \Delta^+(R)$.

[0063] There are several codeword schemes that satisfy our requirements. One such scheme is parity encoding in which the size of each codeword is one word. The parity code is the bitwise exclusive-or of the words in the region. Thus the i 'th bit of the codeword represents the parity of the i 'th bit of each word on the region. For the case of parity, $\oplus$ and Δ are all the same—they compute bitwise exclusive-or. It is easy to check that the parity code has the properties that we require. In particular, $\Delta(M)$ for an update M simply consists of the exclusive-or of the before-update and after-update values of words involved in the update. Also, $\Delta(M)$ requires only one word of storage.

[0064] There are other choices for error-detection codes, for example a summation scheme where the codeword is the word-wise summation of all data on the region modulo 2^{32} . Any such codeword scheme satisfying our requirements can be used. Our focus here is not on the choice of the best error detecting code, but rather on the integration of such a code with a transaction processing system.

[0065] Schemes for Prevention, Detection and Recovery

[0066] In this section, we briefly outline the schemes we will present for the respective levels of protection described above.

[0067] We begin with schemes which prevent corruption. The Hardware Memory Protection scheme may be used to prevent direct corruption. Furthermore, since the database is not corrupted, no explicit recovery is required. If, however, the performance hit of hardware protection is unacceptable, an alternative is to detect direct physical corruption, while preventing indirect physical corruption. This may be accomplished with the codeword-based scheme of Read Prechecking, in which codewords are associated with regions of the database, and these codewords are checked against the actual contents prior to every read.

[0068] Direct physical corruption may be detected rather than prevented with this same codeword arrangement by running periodic audits of the protected data. Based on the performance of Read Prechecking, we recommend using Read Logging to detect transaction-carried corruption. In fact, the use of read logging opens up several new possibilities in error detection and recovery. In particular, if transaction-carried corruption has occurred the read logs provide a way of tracing history and detecting the transactions that were affected by the corruption. Because this

ability to trace affected transaction extends to logical corruption, this technique aids in recovery from errors against which the previously known techniques for corruption prevention are ineffective.

[0069] Database Model

[0070] FIG. 2 provides an overview of the structures used for recovery. Our corruption detection and recovery algorithms are expressed in terms of the Dali recovery algorithm. The Dali recovery algorithm provides very general support for high concurrency, multilevel operations and minimal interference with transaction processing in a main-memory database system. More details can be found in our companion applications and our article describing the Dali main memory manager, "The Architecture of the Dali Main-Memory Storage Manager, referred to above and incorporated herein by reference.

[0071] Multi-level Recovery: Dali implements a main-memory version of multi-level recovery. A multi-level transaction processing system consists of n logical levels of abstraction, with operations at each level invoking operations at lower levels. Transactions themselves are modeled as operations at level n , with level 0 consisting of physical updates. Multi-level recovery permits some locks to be released early to increase concurrency. For example, many systems have two levels defined below the transaction level: a tuple level and a page level. Locks on pages are released early, but locks on tuples are held for transaction duration.

[0072] Latching Protocol: In Dali, the extent of physical latching used to protect the lowest level physical updates is left to the database implementor. (Often these physical updates are covered by higher level locks, leading to efficient implementations of concurrency control for main-memory.) Thus, we merely assume that a physical update occurs on a locking region (not to be confused with a protection region, though they may be the same in a given system), which may be a physical space like a page, or a logical space like a linked list or tree. These lowest level updates are assumed to be covered by a region lock.

[0073] Undo and Redo Logging: Updates in Dali are done in-place, and updates by a transaction must be bracketed by calls to the functions beginUpdate and endUpdate. Each physical update to a database region generates an undo to part of a locking region image and a redo image for use in transaction abort and crash recovery. Undo and redo logs in Dali are stored on a per-transaction basis (local logging). When a lower-level operation is committed, the redo log records are moved from the local redo log to the system log tail in memory, and the undo information for that operation is replaced with a logical undo record. Both steps take place prior to the release of lower level locks. A copy of the logical undo description is included in the operation commit log record for use in restart recovery. The Dali recovery algorithm repeats history on a physical level, so during undo processing, redo records are generated as for forward processing.

[0074] Log Flush: The contents of the system log tail are flushed to the stable system log on disk when a transaction commits, or during a checkpoint. The sys-

tem log latch must be obtained before performing a flush, so that flushes do not occur concurrently. The stable system log and the tail are together called the system log. The variable end_of_stable_log stores a pointer into the system log such that all records prior to the pointer are known to have been flushed to the stable system log. While flushing physical log records, we also note which pages were written ("dirtied") by the log record. This information about dirty pages is noted in the dirty page table (dpt).

[0075] Logical Undo: All redo actions are physical, but when an operation commits, an operation commit log record is added to the redo log, containing logical undo description for that operation. These records are used so that at system recovery, logical undo information is available for all committed operations whose enclosing operation (which may be the transaction itself) has not committed. For transaction rollback during normal execution, the corresponding undo records in the transaction's local undo log are used instead.

[0076] Due to the requirements of multi-level recovery, at any point in time a transaction's undo log consists of some number of logical undo actions followed by some number of physical undo actions. If a transaction aborts, it executes the undo log in reverse order.

[0077] Transaction Rollback: When a transaction is rolled back during normal operation its local undo log is traversed in reverse order, and the undo descriptions are used to undo operations of the transaction. In the case of physical update operations, the undo is done physically, while in the case of logical operations, the undo is done by executing the undo operation.

[0078] Following the philosophy of repeating history, both these actions generate redo logs representing the physical updates (taken on their behalf. Additionally, each undo operation also generates redo log records to note the begin and commit of the undo operation, just like a regular operation. At the end of rollback a transaction abort record is written out, indicating that they are an action taken on behalf of an undo log record.

[0079] Checkpoints: Again, FIG. 2 provides an overview of the structures used for recovery. The database is mapped into the address space of each process. During a checkpoint, dirty pages from the in-memory database image are written to disk. In fact, two checkpoint images Ckpt_A and Ckpt_B are stored on disk, as is cur_ckpt, an "anchor" pointing to the most recent valid checkpoint image for the database. During subsequent checkpoints, the newly dirty portions of the database are written alternately to the two checkpoint images (this is called ping-pong checkpointing). The anchor is switched to point to the new checkpoint only after checkpointing has been successfully completed.

[0080] Thus, even if one checkpoint image is corrupted (due to writing corrupted data during checkpointing or due to failure during checkpointing) the other checkpoint image is still available for recovery.

[0081] Information about active transactions is stored in an active transaction table, referred to herein as the ATT. Due to local logging, the entry for each transaction in the ATT contains local undo and redo logs. In addition to the

database image, a copy of the ATT with the local undo logs and a copy of the dirty page table (dpt) are stored with each checkpoint.

[0082] Note that physical undo information is moved to disk only during a checkpoint. The undo information is taken by the checkpointer directly from the local undo logs of each transaction. (Thus, physical undo log records are never written to disk for transactions which take place between checkpoints.)

[0083] Restart Recovery: Restart recovery starts from the last completed checkpoint image, and replays all redo logs (repeating history). When the end of log is reached, incomplete transactions (those without operation commit or abort records) are rolled back, using the logical undo information stored in either the ATT or operation commit log records. Undo information is available in the redo log for all operations at level 1 and higher. For level 0 (physical updates) the undo information is available in the checkpointed undo log. (Or is reconstructed from the checkpoint image during redo.)

[0084] Due to multi-level recovery, the rollback is done level by level, with all incomplete operations at level i being rolled back before any incomplete actions at level $i+1$ are rolled back.

[0085] Preventing Corruption

[0086] The prevention of physical corruption can take place at two times—at the time of the direct corruption (the “bad write”), or at the time of an attempt to read the corrupt data (indirect corruption). In this section, we present the Hardware Protection scheme which takes the first approach, and the Read Prechecking scheme which takes the second.

[0087] Hardware Protection

[0088] Direct physical corruption can be largely prevented by using virtual memory support provided by most operating systems. The basic approach is to un-protect a page before an access, and re-protect it afterwards. The un-protection and re-protection are done in the database code that surrounds an update, for example, fixing and unfixing a page in the buffer pool. Thus, if an update is done outside of these routines, the page is very likely to be protected, and the hardware mechanism will cause the program to be terminated. If database processes are threaded, however, un-protected pages will be vulnerable to other threads, as hardware memory protection is a per-process resource.

[0089] We note that greater speed with less safety may be available if the page is left un-protected for a period of time—until the end of the enclosing operation, or transaction.

[0090] Turning protection on and off as above can be very expensive since each protect/un-protect involves a system call (for security reasons), which on most current generation operating systems continues to be very expensive (around 20,000 instructions). For example, on a SPARCStation 20 Model 50, only about 16,000 protect/un-protect pairs can be performed in a second.

[0091] While relatively good performance has been obtained, we specially modified a research operating system to take additional advantage of hardware features. We are interested in solutions which are applicable to today's

DBMS on a wide variety of standard hardware and operating systems. Unfortunately, system calls and in particular memory protection and un-protection are expensive on these systems, and the problem does not seem likely to get better.

[0092] Read Prechecking

[0093] An alternative to preventing direct corruption of data is preventing the-use of that corrupted data by a transaction. To accomplish this, codewords are maintained for page-sized or smaller regions of data known as protection regions. When data in the region is updated through the prescribed interface, the codeword is updated along with the data. By using word-wise parity, or a similar scheme, updates to the codeword can be made based only on the portion of the protection region which was actually updated—the remainder of the region need not be read. Selection of an encoding and parity protection scheme may be accomplished in any manner known to those in the art appropriate.

[0094] In the Read Prechecking scheme, the consistency between the data in a protection region and its codeword is checked during each read of persistent data. This scheme and the Data Codeword scheme for direct corruption detection, described below, both use the same codewords maintained during application updates.

[0095] FIG. 3 shows a database memory 300 comprising a plurality of pages 301-1 to 301-3 and codewords for each page, for example, 321-2 for page 301-2, with a latch 311-2 for processes 331-1331-2 to 331-n wanting access to a page 301-2 or a codeword 321-2 for that page useful in describing a simple read prechecking scheme. A protection latch, for example, 311-2 is associated with each protection region and acquired exclusively when data is being updated, or when a reader needs to check the region against the codeword. A shared protection latch is associated with each protection region so that, by holding the latch in exclusive mode, the reader can obtain an update consistent image of the region and the codeword. Updates to data in the region hold this latch 311-2 in shared mode, so that updates to different portions of the region may go on concurrently. Each codeword has an associated codeword latch, which is used to serialize updates to the codeword. More than one codeword can share a single latch; the loss in concurrency is not much since codeword updates take very little time, and thus the latch is acquired only for very short durations.

[0096] A flag, for example, with the name codeword-applied, is stored in the undo log record for a physical update to indicate whether the change to the codeword has been applied corresponding to the update. This flag allows transaction rollback between a beginUpdate and an endUpdate, the functions which, in Dali, bracket a physical update.

[0097] We now present the actions taken during normal transaction processing. In the description of steps taken for update and initialization processing, we assume these actions are taken for an update M , made up of an undo U and redo R . For abort processing, M^{-1} is used to represent the inverse update which restores the data to its original value U .

[0098] Begin Update: At the beginning of an update, the protection latch for the region is acquired, the undo image of the update is noted in the undo log, and the flag codeword-applied in the undo log record is set to false.

[0099] End Update: When an update is completed, the redo image is recorded from the updated region, and the undo image in the undo log record is used with this image to determine the change in the codeword. The codeword is updated with this change, and the operator $\oplus$ is used to compute $\Delta(M)$ from the undo image U and redo image R in the log.

[0100] The codeword latch is acquired in exclusive mode, and the codeword, C, is changed to $C \oplus \Delta(M)$ and the flag "codeword-applied" in the undo log record for the update is set to true. Finally, the protection latch is released.

[0101] Undo Update: When undo processing executes logical actions, the Dali recovery algorithm generates redo log records for updates, just as during normal processing. Correspondingly, the codeword is changed to reflect each update, just as during forward processing. When undo processing encounters a physical undo log record, then it must be handled differently based on whether redo processing had taken place, as represented by the flag codeword-applied in the undo log record. If this flag is set to true, the protection latch is acquired and the codeword for the protection region is modified to back out the update. If the flag is false, no change is made to the codeword, and the protection latch need not be acquired, as it is already held from forward processing. Regardless of the value of the flag, the other undo actions, such as applying the update and generating a log record for the undo, are executed. Finally, the protection latch is released.

[0102] Read: To verify the integrity of the data in the protection region, the reader acquires the protection latch for the region, computes the codeword value of the region and compares this with the stored codeword for that region.

[0103] In order to prevent corruption from reaching the disk image, the checkpoint performs the same integrity check as the reader. If either a reader or the checkpoint finds that a region does not match the codeword for that region, steps are taken as will be described herein to recover the data from this corruption. Note that since corruption cannot be propagated, recovery under the cache recovery model is sufficient.

[0104] Read Prechecking in ARIES

[0105] In this section, we describe how the Read Precheck scheme would need to be modified to work on a page-based architecture such as the ARIES system. In ARIES or other page-based systems, the protection region may be chosen to be the page, in which case the page latch can be used as the protection latch. However our performance study indicates that this may not lead to acceptable performance, thus it may be necessary to associate multiple codewords with a given page. The codeword is associated either with the page, for example, if the codeword will be used to check the integrity of disk writes and reads or with the buffer control block (13CB) if otherwise. The "codeword-applied" flag is stored in the BCB. This flag is needed in case rollback occurs between the FixForUpdate and UnFix calls. If rollback cannot occur in this interval, this flag can be dispensed with and the steps below should behave as if it is always set. As mentioned earlier we assume only one update occurs within

a FixForUpdate and UnFix pair. Our scheme can be extended easily to allow multiple updates, by tracking a region which is a union of the individual regions, and eliminating overlaps.

[0106] The actions taken at various points during an update are described below.

[0107] FixForUpdate: First the page is latched in exclusive mode to prevent any concurrent updates on the page or to the codeword of the page. The undo image of the update is then noted in the undo log, and the flag codeword-applied is set to false. At this time, the codeword is updated with $\Delta^-(U)$ (that is, the Δ for an update which changes the value from U to all zeros) and the region to be updated is noted in the BCB (by storing its offset and length). If the update is physiological (that is, log records that identify a page physically, but record the update to be performed logically as an operation) then, if it is possible to determine an area within the page that may be modified, the offset and length of the area may be stored in the BCB. All updates by the physiological operation must then be within this area. If at this time it is not possible to determine the area affected by the update, the area to be updated is simply assumed to be the entire page.

[0108] Let us define M as an update consisting of old and new values of the updated part of a protected region; then, the effect on the codeword for the region may be defined as ΔM . Then, if we treat M as if it is composed of two updates, one from the initial value of the updated area U to a value of all zeros, and a second from all zeros to the value R it holds after M, then we define $\Delta^-(U)$ as the undo value and $\Delta^+(R)$ as the redo value for the update from all zeros to R.

[0109] UnFix: At the UnFix call, if the original fix was for an update, then the updated region R is determined from the region noted in the BCB, and the codeword for the page is updated with $\Delta^+(R)$ (that is, the Δ for an update which changes the value from all zeros to R). The flag codeword-applied is set. Finally the latch on the page is released.

[0110] Undo Update: When undo processing executes logical actions, all codeword updates are done just as for forward processing. When undo processing encounters a physical/physiological undo log record, it must be handled differently based on whether the codeword has been updated during UnFix or not. To do so, if the flag codeword-applied is not set in the BCB, then the abort was called between the FixForUpdate and UnFix. In this case, the update to area during undo is taken to be from all zeros to the old value; thus the codeword update $\Delta^-(U)$ for the restored value U of the affected area is used to update the codeword for the page. In addition the flag "codeword-applied" is set. If the flag codeword-applied was already set, the update is taken to be from the current value to the old value, and the codeword for the page is correspondingly updated.

[0111] Page Steal: In order to write out a page, the page latch is acquired in shared mode, and the codeword is computed for the page. This computed codeword is compared with the codeword stored for

the page in the BCB, and if they are not equal, then the data in the page has been corrupted. Appropriate corruption recovery actions are then taken, as will be described below.

[0112] Detecting Corruption

[0113] In this section, we describe the Data Codeword scheme for detecting direct physical corruption and the Read Logging scheme for detecting transaction-carried physical and logical corruption.

[0114] Data Codeword

[0115] Detecting (but not preventing) direct physical corruption can be accomplished with a variant of the Read Prechecking scheme described above. The maintenance of the codewords is accomplished in the same manner, however, the check of the codeword on each read is dropped in favor of periodic audits.

[0116] Since transaction-carried physical corruption is possible in this scheme, additional care must be taken during checkpointing to ensure that an uncorrupted image exists on disk. The process of auditing is nothing more than an asynchronous check of consistency between the contents of a protection region and the codeword for that region. This can be carried out just as if a read of the region were taking place in the Read Prechecking scheme.

[0117] Since prechecks are not being performed, and audits are asynchronous, it makes sense to use significantly larger protection regions. In this case the protection latch may become a concurrency bottleneck. If so, a new latch, the codeword latch, may be introduced to guard the update to the actual codewords, and the protection latch need only be held in shared mode by updaters. During audit, the protection latch must be taken in exclusive mode to obtain a consistent image of the protection region and associated codeword. In particular, data is audited during the propagation to disk by the checkpointing (or page-steal in a page-based system).

[0118] Note that this scheme by itself supports only the cache recovery model, thus indirect corruption is not prevented, rather one attempts to audit frequently enough to repair direct physical corruption before it is encountered.

[0119] Corruption Detection with Deferred Maintenance

[0120] We now describe an alternate scheme for physical error detection in which the maintenance of codewords is deferred to improve concurrency. This scheme stores codeword updates in log records and updates codewords, for example, during a log flush rather than during a data update (or immediately as described above). Updaters need not obtain any page latches, and checkpointing, codeword-auditing and page-flushing do not interfere with updates. FIG. 4A shows a database memory 400 organized with codewords 421 for page regions having a page portion protection latch 441a permitting only one process to read or modify a codeword at a time useful in describing deferred codeword maintenance. Since codeword updates are done during log flush, the system-log latch 441, which prevents concurrent flushes, serves to serialize access to the codewords. Read processes can go in at any time to read. Also, latch 441 allows several processes to modify a page concurrently, but only one process can go in to verify a page. Thus, the deferred maintenance scheme can result in

increased throughput for update transactions, especially in main-memory databases, where the relative cost of latching can be substantial. Other locks/latches 411-1, 411-2 . . . 411-N in the system control access to individual parts of the page 401-1.

[0121] For this scheme, each redo log record has an additional field, the "redo delta", which restores the value $\Delta(M)$, where M is the update denoted by the log record. Also, each undo record has a codeword-applied flag. We assume that the area covered by a redo log record does not span two or more protected regions; the assumption can be easily assured by splitting log records that span protection boundaries into multiple pieces.

[0122] The actions taken at various steps are described below and are in addition to the regular actions taken during these steps.

[0123] Begin Update: Add an undo log record for the update to the local undo log of the transaction that performed the update, and set codeword-applied in the undo log record to false.

[0124] End Update: Create a redo log record for the update. Find the undo log record for the update, and find the updated area from the undo log. Compute $\Delta(M)$ from the undo image in the undo log and the new value of the updated area, and store it in the redo Δ field in the redo log record. Add the redo log for the update to the redo log, and set the codeword-applied flag in the undo log record to true.

[0125] Undo Update: For each physical update in the undo log, a proxy redo log record is generated, whose redo image is the old value from the undo record. If the codeword-applied flag for the undo record is true, then a redo record has already been generated with $\Delta(M)$, so $\Delta(M^{-1})$ must be included in the proxy record to reverse the effect. $\Delta(M^{-1})$ is computed from the current contents of the region and the undo log record, and stored in the redo Δ of the proxy log record.

[0126] If the codeword-applied flag is false, then no redo log record has been generated, and thus no codeword delta will be applied by the flusher process. Since the codeword has not been changed and should not be changed, the proxy record is created as usual and 0 (a special value that results in no change to the codeword) is stored in its redo Δ .

[0127] In either case, the physical undo log record is processed by replacing the image in the database with the undo image in the log. Note that the logical undo actions generate physical updates; codeword processing is done for these updates as described above.

[0128] Flush: When flushing a physical redo log record, the flusher applies redo $\Delta(M)$ from the log record to the codeword for that page. Note that the system log latch is held for the duration of the flush. Also, the variable end_of_stable_log is updated to reflect the amount of log which is known to have made it to disk.

[0129] Auditing

[0130] An algorithm for performing an audit under the deferred maintenance scheme will now be described with

reference to FIG. 4A. A database is shown in dashed line with some pages 470, 475, and 478 identified and their corresponding codewords 471, 476 and 479 of a codeword table 430. Clients attempt to read/write and maintain client personal logs 440-1, 440-2 and a log for client n not called out. A flusher process to disc is shown as flusher process 460 and a flush latch 461. A checkpoint image 450 is maintained for changes to pages of the database made by the clients.

[0131] In the logging-based scheme, while the updater's task is simpler, the job of the auditor has become significantly more difficult. When an auditor algorithm reads the page, it does so fuzzily, and partial updates may be captured. Log information must then be used to bring the page to an update-consistent state, so that it can be compared with a codeword value. This requires performing a small-scale version of the Dali recovery algorithm at checkpoint time. To avoid the expense of executing this algorithm for the entire database we introduce a fuzzy precheck. The idea of the precheck is simply to compute the codeword value of the page, ignoring ongoing updates, and compare that to the codeword value in the codeword table. If the two match, we assume that the page is correct. Note that this introduces a slightly increased risk of missing an error, because a valid, in-progress set of updates which have not made it to the table might exactly match the effect in the table of a corruption. However, we consider the probability of this to be approximately the same as the probability of an error going undetected by the codeword which is dependent on the codeword scheme, but is extremely small. We store the set of pages which fail this precheck test in the set AU_needed.

[0132] We now present the steps taken to audit the database:

[0133] 1. For each page

[0134] (a) Note the value in the codeword table 430 for the page, for example, page codeword value 471 for page 470 or codeword value 476 for page 475.

[0135] (b) Compute its codeword (without latches or locks).

[0136] (c) Note the value, for example, value 471, in the codeword table 430 for the page, for example, page 470, again.

[0137] (d) If the computed value does not match either noted value, add the page, for example, page 470, to AU_needed.

[0138] 2. Note end_of_stable_log into AU_begin.

[0139] 3. Copy pages in AU_needed to the side, for example area 450.

[0140] 4. Extract the trailing physical undo records for in-process transactions from the active transaction table ATT. Call this collection of physical records the AU_att. Records are gathered from different transactions independently, using a latch on the entry to ensure that operations are not committed by a transaction while we are gathering its log.

[0141] 5. Get flush latch 461, and execute a flush to cause codewords from outstanding log records to be applied to the codeword table 430. Note the new end_of_stable_log in AU_end. Note the codeword

values for all pages in AU_needed into a copy of the codeword table called AU_codewords. Finally, release the flush latch 461.

[0142] 6. Scan the system log from AU_begin to AU_end. Physical redo records which apply to pages in AU_needed are applied to the side copy 450 of those pages. Also, if the undo corresponding to this physical redo is in the AU_att, it is removed.

[0143] 7. All remaining physical undo records from AU_att which affect pages in AU_needed are applied to the checkpoint image 450.

[0144] 8. Compute codewords of each page in AU_needed, and compare to the value in AU_codewords. If any differ, report that basic corruption has occurred on this page.

[0145] At the end of the flush in step 5, a certain set of updates for each page has been applied to the codeword table 430, and this is captured in AU_codewords. The purpose of the rest of the algorithm through step 7 is to ensure that the side copy of each page being checked contains exactly those updates. All prior updates which may have been partially or completely missing from the checkpoint image 450 are reapplied by step 7. Since in Dali redo log records are not immediately placed in the system log, some updates may have taken place which are only recorded in transaction's local redo and undo logs. Note that the codeword delta for these updates is not reflected in the codeword table 430. Therefore, these updates are undone in step 7, leaving the side image of a page consistent with the updates reflected in the codeword for the page in AU_codewords.

[0146] In this algorithm, individual transactions are blocked long enough to gather the outstanding physical undo records for the current operation.

[0147] Read Logging

[0148] In order to detect indirect physical and logical corruption, we introduce the idea of limited read logging. When a data item is read (the data item could be a tuple, an index node or any persistent data structure required for database consistency), a read log record identifying the start point and the length of the data that was read is written out. Of course, equivalent schemes may come to mind of one of ordinary skill in the art such as logging an end point and length of data read or other scheme. Since most writes are preceded by reads of the same data, one may assume that a write of a region is implicitly a read also, and thus significantly reduce the number of read log records required. Should an audit indicate that certain data is corrupt, the read log records can help determine if a transaction has in fact read the corrupt data. We shall see that the read log records provide a mechanism for tracing the flow of indirect corruption in the database and thus determining the set of transactions affected by an instance of direct corruption.

[0149] In database applications, a write log record is maintained of writes to a database. In a preferred embodiment of the present invention, the write log record is combined with the presently suggested read log record as a combined record. In an alternative embodiment, the read log record may be maintained separately from the write log record with a possible loss of efficiency in the recovery algorithms.

[0150] Optionally, a codeword for the data that was read can also be written out with the read log record. In the event of a crash, the codeword can be used to detect any corruption which may have occurred since the last audit, which would not otherwise be detected. The codeword is also used to detect when exactly an item was physically corrupted, given that a corruption has been detected by some other means (either an audit or by external means). For example, when an audit detects a corrupted protection region it must be assumed, in the absence of a codeword in the read log record, that any transaction reading the region since the time of the previous successful audit read the corrupted data. With the codeword, a precise check can be made.

[0151] Read Logging, unlike the other techniques presented so far, provides the benefits of being able to detect propagation of logical corruption. When read logging is used for this purpose, differences detected at the physical level may overestimate the flow of corruption. In this case, operations may log a checksum of the logical state found. How exactly to do so is dependent on the operation. However, the following property must be satisfied—if the checksums are the same (with a high probability) the two operation executions gave the same logical result. For example, an index lookup would log a checksum of the index entries that it retrieved, ignoring physical details such as location in the index.

[0152] There are two alternative ways of creating the read log record; the system designer must choose and use only one of them:

[0153] 1) Physical read logs, which specify the start point and the length of the data that was read. Additionally the records can store a checksum of the data that was read, used during corruption recovery to detect reads of corrupted data.

[0154] 2) Logical read logs, which use lock information for read logging. When a transaction locks a data item, a log record noting the lock information (name of the item and type of lock) is written out. This is done for both read and update locks, so that it is possible to detect which transaction has read information written by which other transaction. In the absence of write lock information, the physical redo log records alone may not be sufficient to infer which transactions read items written by a given transaction.

[0155] Generating Checkpoints Free of Corruption

[0156] Since Read Logging supports recovery from indirect corruption, it becomes crucial that the disk image be free not only of direct corruption, but indirect corruption as well, so that recovery does not require loading an archive image. Thus, when propagating dirty pages from memory to disk, it is not sufficient to audit the pages being written. Even if none of the dirty pages has direct physical corruption, it is possible that a "clean" page has direct corruption, and a transaction carried this corruption over to a page that was written out. Thus the checkpoint would have data that is indirectly corrupted.

[0157] With deferred maintenance of codewords or read logging, the correct way of ensuring that the checkpoint is free of corruption is to create the checkpoint, and after the checkpoint has been written out, audit every page in the database. If no page in the database has direct corruption, no indirect corruption could have occurred either. We can then certify the checkpoint free of corruption.

[0158] This technique cannot be directly applied to page flushes in a disk-based system, since it amounts to auditing all pages in the buffer cache before any write of a dirty page to disk (at page steal time). However, a similar strategy can be followed if a set of pages are copied to the side, and then an audit of all pages is performed before writing them to the database image on disk. To ensure that direct physical corruption does not escape undetected, a clean page which is being discarded to make room for a new page must also be audited.

[0159] Corruption Spread

[0160] Unlike prevention, the success of corruption detection and recovery is dependent on the rate at which corruption spreads and how quickly it is detected. The speed at which direct corruption is detected depends on the frequency of audits in the system, unless Read Prechecking is used, in which case the corruption may also be detected by a failed precheck. The rate of corruption spread may not be easily quantified, as it depends on two factors which are dependent on details of DBMS implementation and a particular application's use of the database.

[0161] First is the probability that an initial corrupt read will take place in a given time period. The second is the probability that a new transaction will read data corrupted indirectly by previous transactions. The probability of an initial corrupt read varies based on the application workload, internal DBMS implementation techniques, and which data is directly corrupted. Certainly, there is some data in any database that will corrupt almost any transaction that follows, for example, a key or pointer in the root node of a tree index. Similarly, there may be data that will cause no damage, such as currently free space, and data which will cause little damage, such as a text description which is not used in processing, just for display.

[0162] The second factor, the probability that a new transaction will read data which is already corrupted, depends on the frequency of contact between transactions, which again depends on the application and DBMS implementation. In addition, this probability will grow over time as more data is indirectly corrupted. Consider two variants of an application which updates customer information. In one variant, each transaction updates global summaries of the total business done with all customers, and in the other variant such global statistics are computed when they are needed, and individual transactions touch only one customer. Clearly, corruption would propagate extremely quickly in the first case, and much more slowly in the second.

[0163] In the worst case, any transaction would touch data written by a recent transaction. Any application in which each transaction both updates and uses a global statistic would exhibit this worst-case behavior. For example, a database which tracked the contents of a warehouse might have a summary of how much space is available, which would be checked for each incoming transaction before looking for space of the right size, etc. At the other extreme, transactions may deal with customer accounts with no overlap between customers (maintaining global statistics as in the previous example can lead to very poor concurrency). In this case, corruption would only spread outside of a

particular customer record through DBMS structures, such as index nodes and free space lists. While corruption could still spread, the spread would be much slower.

[0164] In summary, the total amount of corruption allowed to arise in the database is determined by the speed at which the corruption spreads in a particular application/DBMS pair, and the frequency with which the data is audited.

[0165] Corruption Recovery

[0166] So far we have seen how to detect direct physical corruption by means of codeword schemes, and to log information about reads to detect indirect physical and logical corruption. We now consider how to recover from corruption once it is detected. Note that these algorithms for recovery from corruption are tightly integrated with restart recovery. On detecting an error, we simply note the region or regions failing the audit, and cause the database to crash. Corruption recovery is handled as part of the subsequent restart recovery.

[0167] Cache Recovery Model

[0168] This algorithm is useful to recover from direct corruption when recovery from indirect corruption is not required. It is invoked when a precheck fails in the Read Prechecking scheme (as indirect corruption could not have occurred), or when an audit detects a codeword error in the Data Codeword scheme (as the Data Codeword scheme does not address indirect corruption). By ensuring that directly corrupted data does not get propagated to the disk image, recovery from a corrupted cache image can be accomplished with standard techniques. Such a technique is known for ARIES for use when a page is latched by a process which was unexpectedly terminated. For Dali, a similar technique can be used to reload the latest checkpoint and replay physical log records forward.

[0169] Prior-State Recovery Model

[0170] The prior-state model can be used to recover from any form of corruption, once the source of that corruption is determined, and the recovery can be accomplished with standard techniques. In this case, the techniques are similar to media failure. An archive image of the database must be loaded, and the log played forward to the latest point known to be prior to the introduction of corruption. Once that point is reached, the log is truncated, and restart recovery is executed (possibly generating additional log records).

[0171] Redo-Transaction Recovery Model

[0172] In order to recover from corruption under the redo-transaction model for corruption recovery, we require that a logical redo log is available along with the transaction code necessary to redo transactions. In particular, the logical log must be at the transaction level, that is the transaction's external inputs (e.g., from the user) are saved. In the following, we assume the transaction writes this logical description with its commit record.

[0173] We shall also require that the commit order be the same as the serialization order; this can be ensured by requiring that transactions hold all locks (or in multi-level recovery schemes, all locks at level $n-1$) till end of transaction.

[0174] Recovery

[0175] Corruption recovery begins as in the prior-state model—the database is recovered to a transaction consistent state known to be prior to the introduction of corruption. The tail of the log not used by this recovery is saved in OldLog before the log is truncated (see Section). The re-executed transactions will read non-corrupted data, and produce non-corrupted results.

[0176] When comparing transactions, we scan the logs in parallel; if we find that they both have operation begin log records indicating the operation has a logical checksum, and the checksums in the operation begin log records are the same, we stop comparing the checksums on the redo log records generated by the operations. When we reach the operation commit on both logs, we compare the logical checksum to see if the operation results were the same.

[0177] Handling External Writes and User Notifications

[0178] Changes in external writes have to be handled manually, and even changes to the database may require notification to the users in some cases. Identifying which transactions were corrupted enables humans to focus on these transactions, thereby reducing the human workload. Detecting what external writes changed as a result of re-execution simply requires external writes to be logged. The log comparison procedure above will then detect changes in external writes.

[0179] User Notification

[0180] Depending on the nature of the corruption, it may be helpful to determine which results exposed to the outside world were corrupt. To accomplish this, any transaction which reports to the user must log the values output to the user. During re-execution, these values can be checked against the values logged by the transaction as it re-executes. If they do not match, the user is notified.

[0181] Extension: Logical Corruption

[0182] The above algorithm can be easily extended to recover from logical corruption. This is accomplished by amending Old_Log, prior to recovery, in order to correct an erroneous transaction or transactions. For instance, we may have fixed errors in the transaction code, or we may use correct user inputs in place of wrong user inputs. We may even delete the transaction or replace it with one or more other transactions. If it is not known when the direct logical corruption occurred, techniques as described below can aid in making this determination.

[0183] Delete-Transaction Model

[0184] If a logical log is not available, recovery may take place under the delete-transaction model discussed above. For our delete-transaction model recovery algorithm, we need a checkpoint which is update-consistent in addition to being free from corruption. However, in Dali, a checkpoint being used for recovery is not necessarily update-consistent until recovery has completed (that is, physical changes may only be partially reflected in the checkpoint image, and certain updates may be present when earlier updates are not).

[0185] The Dali algorithm to obtain an update-consistent checkpoint uses a portion of the redo log and AIT to bring the checkpoint to a consistent state before the anchor is toggled and the checkpoint made active. Once performed, the checkpoint is update-consistent with a point in the log, CK_end.

[0186] We will outline how to get a checkpoint that is update consistent with a specific point CK_end in the redo log, by applying undo logs. Such a checkpoint has some important additional properties:

[0187] 1) Effects of redo log records after CK_end are not reflected in the checkpoint image and

[0188] 2) The checkpoint is codeword consistent as of CK_end.

[0189] As a result, when we replay log records after CK_end, whenever we encounter a read log record the database state will be exactly what the read found when the read log record was generated, unless the database was corrupted. That is, we can identify a point in the log such that all physical updates noted in the log up to the point are reflected in the checkpoint, and none of the physical updates in the log after that point are reflected.

[0190] Such a checkpoint can be obtained by a simple modification of the Dali checkpointing scheme. We create a checkpoint exactly as done in Dali, but then perform some physical redos and undos on the checkpoint image as described below, to make it update-consistent.

[0191] Recovery Algorithm

[0192] The main idea of the following scheme is that corruption is removed from the database by refusing during recovery to perform writes which could have been influenced by corrupt data. In order to do this, the transactions which performed those writes must, at the end of the recovery, appear to have aborted instead of committed. Certain other transactions may also be removed from history (by refusing to perform their writes) in order to ensure that these "corrupt" transactions can be effectively removed, and thus ensuring the final history as executed by the recovery algorithm is delete-consistent with the original execution as discussed above.

[0193] Recovery must start from a database image that is known to be non-corrupt. Note that since errors are only detected during checkpointing or auditing, we may not know exactly when the error occurred; the error may have been propagated through several transactions before being detected. FIGS. 5A and 5B comprise a flowchart of the recovery algorithm for the delete transaction model. The starting point of the algorithm is box 500. The algorithm conservatively assumes that the error occurred immediately after Audit_LSN, the point in the log at which the last clean audit began. Box 505 relates to the repetitive process of reading log records and box 515 asks whether we are at Audit_LSN. If yes, then, we add all data noted as corrupt by the last audit to the CorruptDataTable. If not, we enter a redo phase followed by an undo phase.

[0194] Two tables, a CorruptTransTable and a CorruptDataTable are maintained. A transaction is said to have read corrupt data if the data noted in a read or write log record of that transaction is in the CorruptDataTable.

[0195] Restart recovery consists of the redo phase followed by the undo phase. The redo phase initiates with the question what kind of log record is it at box 520.

[0196] Redo Phase

[0197] The checkpointed database is loaded into memory and the redo phase of the Dali recovery algorithm is initiated, starting the forward log scan from CK_end.

[0198] During the forward scan, the following steps are taken (any log record types not mentioned below are handled as during normal recovery):

[0199] If a read log record or a physical write log record is found, you follow the path in FIG. 5A to box 525, and if this record indicates that the transaction has read corrupted data at box 525, then the transaction is added to CorruptTransTable (where it may already appear) at box 565.

[0200] If a log record for a physical write is found, then there are two cases to consider:

[0201] 1. The transaction that generated the log record is not in CorruptTransTable at box 575: In this case, the redo is applied to the database image as in the Dali recovery algorithm at box 585.

[0202] 2. The transaction that generated the log record is in the CorruptTransTable (CTT): In this case at box 580, the data it would have written is inserted into CorruptDataTable (CDT). However, the data is not updated because the transaction is in the CTT.

[0203] If a begin operation log record is found for a transaction that is not in CorruptTransTable at box 530, then it is checked against the operations in the undo logs of all transactions currently in CorruptTransTable. If it conflicts with one of these operations, then the transaction is added to CorruptTransTable at box 560. This ensures that the earlier corrupt transaction can be rolled back. If it does not conflict, then it is handled as in the normal restart recovery algorithm at box 555.

[0204] If a logical record such as commit operation, commit transaction or abort transaction is found, the record is ignored at box 535 if the transaction that generated the log record is in CorruptTransTable. Otherwise, the record is handled as in normal restart recovery at box 550.

[0205] When Audit_LSN is passed at box 515, all data noted to be corrupt by the last audit or by analysis of a logical error, is added to CorruptDataTable at box 570.

[0206] Undo Phase

[0207] At the end of the forward scan, incomplete transactions are rolled back. following the normal Dali restart recovery algorithms undo phase. As in the normal Dali algorithm, undo of all incomplete transactions is performed logically level by level. Note that at the end of the redo phase, each transaction in CorruptTransTable has a (possibly empty) undo log. This log records the actions taken by the transaction before it first read corrupted data. During the undo phase at box 590, these portions of the corrupt transactions are being undone along with the transactions which were in progress at the time of the crash.

[0208] Checkpoint

[0209] The extended restart recovery algorithm is completed by performing a checkpoint at box 595 to ensure that recovery following any further crashes will find a clean database free of corruption, and to avoid the insertion of records into the log during the rollback of a corrupt trans-

action. If the checkpoint were not performed, a future recovery may rediscover the same corruption and in fact additionally declare transactions that started after this recovery phase to also be corrupted. Note that this checkpoint invalidates all archives. The log may be amended during recovery to avoid this problem.

[0210] Discussion

[0211] The database image at the end of the above algorithm should reflect a delete history that is consistent (in this case, conflict-consistent) with the original transaction history. To see (informally) that this is the case, first observe that all top-level reads of non-deleted transactions read the same value in the history played during recovery as in the original history. This is because any data that could possibly have been read with different values was previously placed in CorruptDataTable, and top-level reads must be implemented in terms of physical-level reads where corruption is tracked. The second observation is that the database image is consistent and contains the original contents plus the writes of those transactions which do not appear in the delete set. This follows from the correctness of the original recovery algorithm, and the fact that the initial portion of corrupted transactions can be rolled back during the undo phase along with normal incomplete transactions to produce a consistent image. This is ensured since we do not allow any subsequent operations which conflict with these operations to begin.

[0212] Extension: Codewords in Read Log Records

[0213] If codewords are stored in read log records, then detection of indirect corruption becomes more precise. In particular, the CorruptDataTable can be dispensed with, and instead, the definition of reading corrupt data given above is replaced by the following two cases:

[0214] 1. A codeword is stored in a read log record, and it does not match the computed codeword for the corresponding region in the database being recovered.

[0215] 2. A codeword is stored in a write log record (indicating that it should be treated as a read followed by a write) and the codeword does not match the computed codeword for the corresponding region in the database.

[0216] A second benefit of storing codewords in read log records is that it is possible to detect physical corruption which occurred after the last audit but before the crash. More precisely, physically corrupt data will be detected if any transaction read it, since during recovery the codeword for these transactions will not match the database image being recovered. Thus, if codewords are present, the corruption recovery algorithm should be executed not only when an error is detected, but also on every restart.

[0217] Note that the modified algorithm produces a recovery schedule which is view-consistent with the original history, thus not propagating corruption when the corrupt transaction wrote the same data to a data item as it would have had in the delete-history. However, these benefits must be traded against a slight degradation in performance (see performance section below for details).

[0218] Extension: Recovering from Logical Corruption

[0219] The algorithm given above can be easily adapted to recover from logical corruption. This corruption may have occurred before the last checkpoint, thus an archive image that was taken prior to the introduction of the error is restored, and CK_end (the point at which recovery starts) taken from that archive. Audit_LSN is defined to be the latest point in the log such that the corruption is known to have occurred after that point. If this point is not known, backwards analysis techniques discussed below can be used to help locate it. Finally, when the point Audit_LSN is passed during recovery, any data which was directly affected by the corrupting error must be added to CorruptDataTable.

[0220] Logical Corruption

[0221] Approaches to recovering from logical corruption have already been given in the context of the redo-transaction and delete-transaction models. In this section, we discuss some issues involved in determining when logical corruption was introduced, and outline an efficient variant of the redo-transaction recovery scheme for a restricted recovery model.

[0222] The techniques described so far do not consider transactions with faulty code that updated the database erroneously, but through the prescribed interface. Similarly, transactions that were executed incorrectly, for instance with wrong user input, have not been considered so far herein. We call transactions such as the above as erroneous transactions. Of course, it is not possible to prevent erroneous transactions, since users as well as programmers can always make mistakes.

[0223] Detecting that there has been an erroneous transaction execution is quite non-trivial, and cannot be done automatically by the system, since the differences between good and erroneous transaction executions are at a semantic level above what the database may be aware of. Integrity constraints may help detect some such errors, but cannot detect all.

[0224] We assume that humans have somehow become aware that an error has occurred. For instance, a customer may complain of a wrong balance, or an operator may discover that a transaction he ran yesterday ought not to have been executed.

[0225] We have two models of error detection here: where it is known exactly which transaction originally caused the error, or which was the first erroneous transaction. In this case, only forward recovery is required. For example, the case where the operator realizes a transaction he ran ought not to have been executed.

[0226] Once the cause of the error is known, we can find the latest consistent checkpoint (archive image) prior to executing the erroneous transaction, and perform recovery using one of the algorithms given in our corruption recovery section.

[0227] If recovering under the redo-transaction model, however, we do not simply re-execute erroneous transactions, but re-execute a corrected version of the transaction. For instance we may have fixed errors in the transaction code, or we may use correct user inputs in place of wrong user inputs. We may even delete the transaction or replace it with one or more other transactions.

[0228] Backward Analysis—Determining the Source of Corruption

[0229] Even if it is known that logical corruption has occurred, it may not be known which transaction initially introduced the error. For example, a customer has detected a wrong balance but does not know what caused it. Here, two steps are required: detecting the root cause, and then forward recovery from there.

[0230] The database system cannot automatically detect the root cause, but can provide support to humans to detect the problem by supporting backward analysis of the log using the read log records.

[0231] The following algorithm is applicable when certain data are known to be corrupt, and the source (originating transaction) for the corruption is sought. This algorithm assumes that the database log has been enhanced with logs of data items read using the read logging techniques described earlier. The database log contains a record of both writes and reads, and each write is assumed to imply a prior read of the item. The idea is to trace backward through the log, tracking how corruption could have flowed into the known-corrupt data items.

[0232] Assume there are n data items known to be corrupt. Let SuspectData be a set of data suspected of being corrupt. Associate with each suspect data item, D , in SuspectData a set of integers from $1 \dots n$, $A(D)$ which represents the known-corrupt data which could have been affected by corruption if D itself were found to be corrupt. Let SuspectTrans be a set of suspect transactions, each with a set $A(T)$ associated with semantics analogous to $A(D)$: if i is in $A(T)$, then if transaction T has read corrupt data or was itself the source of direct logical corruption, then the known corruption of data item i could be explained as resulting from this error. The goal is to find a single transaction which could explain all the known corruption, that is all the items $1 \dots n$ are in the set $A(T)$ for that transaction.

[0233] In our algorithm, we would process the log backwards; that is, for each log record, do:

[0234] If the log record is a write of data item D by transaction T and if D is in SuspectData, add T to SuspectTrans (if not already present), and set $A(T)$ equal to $A(D) \cup A(T)$.

[0235] If the log record is a read of data item D by transaction T and if T is in SuspectTrans, add D to SuspectData (if not already present) and set $A(D)$ to be $A(D) \cup A(T)$.

[0236] If any $A(T)$ contains all elements from $1 \dots n$, offer transaction T to the user as a possible root cause of the corruption. If the user accepts, done, else continue.

[0237] Now we will discuss an example where the corruption detection and recovery algorithms of the present invention may be applied to advantage. Consider an example where five customers of a bank, Bank A, call over the course of a day and complain that their balance is incorrect. A corruption detection algorithm according to the present invention will proceed by processing the log backwards from the latest time when a bad balance is reported. It may soon be found that a transaction which added interest to accounts had recently updated all five accounts. This

transaction has an associated set, $A(T)$, which contains all five known corrupt data items, and the transaction will be presented to a user of the system as a potential source of corruption. The user upon examining the transaction may determine, however, that the interest rate was computed correctly, and thus prompt the algorithm to continue searching for an explanation for the corruption.

[0238] Subsequently, it may be found that earlier in the log of events, updates to the third and fifth complaining customers' balances had been made due to an ATM (automatic teller machine) withdrawal at Bank B. Since the transaction that posted these withdrawals would read the main data record for Bank B, that record would be added to the SuspectData set. If subsequently in processing (earlier in time) it is discovered that the other complaining customers' accounts had been subject to withdrawals from ATM's owned by the same Bank B, which also read the main data record for Bank B, then the $A(D)$ associated with the main data record for Bank B (D), would contain all known corrupt data items. The next (earlier) transaction to update the main data record for Bank B would be presented to the user as a potential source of corruption. In this example, it may turn out that the amount of money charged by Bank B as an overhead for using their ATM had been incorrectly entered into the database, resulting in the eventual incorrect balances. A user may use any one of the corruption recovery algorithms described herein to help determine which other customers are affected by this incorrect update, as well as other data items which may be affected indirectly such as payments of these charges to Bank B.

[0239] As written, the detection algorithm assumes that all the known corruption originated from a single source. However, the algorithm can be easily modified to consider multiple sources of corruption. For example, it could present to the user whenever two transactions together explained all the corruption, that is the union of $A(T)$ and $A(T1)$ contained all the data elements $1 \dots n$, where T and $T1$ are two transaction in SuspectTrans table.

[0240] Optimizing Recovery in the Redo-Transaction Model

[0241] Unfortunately, under the redo-transaction model of corruption recovery, the logical re-execution of a transaction may take time similar to its original execution; if an error is discovered after several days, it can reasonably be expected to take several days to recover the database. During this time the database would be unavailable.

[0242] Referring to FIG. 6, we now outline an alternative approach to implementing the redo-transaction model which assumes a two-level recovery model with logical or physiological redo logging. Thus, record-level operations are logged, record-level locks are held for the duration of the transaction, and latches are held on pages for the duration of an operation. The goal of the algorithm is to use primarily the log for recovery, only re-executing a select few transactions.

[0243] In the algorithm given below, we may use a CorruptDataTable as in the delete-transaction model recovery algorithm, or use logical read logging as described in above. A transaction is said to read corrupted data if

- [0244] It reads data in the CorruptDataTable, if such a table is used.
- [0245] The logical codeword is computed for an operation during recovery, and it does not match a codeword recorded in the log.
- [0246] In addition to the actions normally taken during recovery, the corruption recovery algorithm proceeds by processing records as follows:
- [0247] Save log records per step 601 for a transaction until a commit or an abort for the transaction is seen.
- [0248] If the commit log record for a transaction is encountered via path 605, then
- [0249] 1) The read log records for the transaction are scanned at step 610 to determine if the transaction has read corrupted data. If so, the transaction is marked as corrupt at step 615.
- [0250] 2) If the transaction is marked as corrupt, it is re-executed logically, and the new logical redo records are used to replace its redo records in the log at step 620. Any data which the original transaction wrote, or was written during the re-execution, is added to the CorruptDataTable, if one is used.
- [0251] 3) If the transaction is not marked as corrupt, its log records are executed at step 625.
- [0252] 4) If an abort record for a transaction is found at step 630, the log records for that transaction are discarded at step 635.
- [0253] Since the redo records are logical and record-level locks are held to the end of transaction, executing the log records at the point where the commit record appears is correct. Note that when a transaction is re-executed, it could generate different log records. In fact, the operations it performs may be completely different from the ones it originally performed. Since transactions are executed at their commit point, this new transaction will serialize with respect to other transactions which might have originally executed concurrently with it. It will read the data as written by any transaction which serialized before it, and if its actions cause any new conflicts with transactions that serialize after it, these transactions will read corrupt data and be re-executed themselves.
- [0254] Performance
- [0255] The goal of our performance study was to compare the relative cost of different levels of protection, for example detection versus prevention, as well as comparing different techniques for obtaining the same level of protection. In each case, we are interested in the impact of the scheme on normal processing as opposed to the time taken for recovery. Corruption recovery is expected to be relatively rare, and the time required is highly dependent on the application and workload. The algorithms studied were implemented in the DataBlitz Storage Manager, a storage manager being developed at Bell Labs based on the Dali main memory storage manager.
- [0256] Performance of mprotect
- [0257] Before describing the comparison of schemes, we begin by looking at the relative performance of memory

protection primitives on commonly available UNIX hardware. In the Table below, we evaluate the basic performance of the memory protection feature on a number of hardware platforms locally available to us. In each case, 2000 pages were protected and then unprotected, and this was repeated 50 times. The number reported is the average number of these pairs of operations which were accomplished per second.

[0258] As shown in this table, the UltraSPARC on which our benchmarks were run is significantly faster at memory protection than the other UNIX platforms to which we had access, leading us to expect that hardware protection may fare worse on other platforms.

Performance of Protect/Unprotect	
Platform	pairs/second
SPARCstation 20	15,600
UltraSPARC2	43,000
HP 9000 C110	3,300
SGI Challenge DM	8,200

- [0259] Implementing Read Logging and Prechecking
- [0260] Computation of codeword values for reads takes place in two contexts: as part of a codeword precheck or when computing the codeword to be stored in a read log record. In general, implementation difficulty must be traded against the window of potentially undetected corruption. To minimize the window for undetected corruption, a codeword computation should be made after the read has occurred. If the computation is made before the read, direct corruption could take place in the window between the computation of the codeword and the read itself.
- [0261] In the case of prechecking, if a read is performed, and a write is made based on that read before the codeword for the read is checked, then this write can potentially cause transaction-carried corruption, which the Read Prechecking scheme is designed to prevent. More precisely, such a write can be allowed, but cannot be visible to another transaction, and it must be possible to undo the write physically, as any logical undo based on a corrupt read cannot be trusted.
- [0262] For example, in a page-based system, one approach is to perform the read precheck when a page is Fix'ed in a shared mode, which leaves some window between the computation and the read in which undetected corruption could occur. Alternatively, the precheck can occur at UnFix time, introducing the possibility that a write is issued which is based on the Fix'ed data before the UnFix takes place, in which case provision must be made to physically undo that write if an error is detected at UnFix time.
- [0263] For read logging, the read log record must appear in the log before any subsequent writes so that the recovery algorithms will consider the writes as suspect. In this case, it is simpler to use the codeword at Fix time.
- [0264] A conservative solution, which may have a significant performance cost, is to copy the information to be read into private space and perform the test and subsequent reads on this copy. In the page-based example, this would happen at Fix time.

[0265] To eliminate any window between checking the codeword for some data and then using it, an explicit call to perform the check must be inserted between reads and any writes which are possibly exposed to another transaction. For example, if pages are fixed in read mode for the duration of a complicated action, then some analysis of the code will be required to add the additional calls.

[0266] In our implementation, which is not page based, we assumed that any write was also a read, thus the codeword was checked during beginUpdate calls. We added calls before all reads of persistent data which were not part of a write. In the case in which a codeword is included with the read log record, then these codewords were added to the write log record as well as being added to the read log record. Thus, calls for each read which was not part of a write were added to the portions of the code which is used by our performance tests, including the allocation code, the relation manager, and one index structure, a hash table.

[0267] Workload

[0268] The workload examined is a single process executing TPC-B style transactions. The database consists of four tables, Branch, Teller, Account, and History, each with 100 bytes per record. Our database contained 100,000 accounts, with 10,000 tellers and 1,000 branches. These ratios are higher than in TPC-B, in order to limit CPU caching effects on the smaller tables. The benchmarks were run on an UltraSPARC with two 200 MHZ processors, and 1 gigabyte of memory. All tables are in memory during each run, with logging and checkpointing ensuring recoverability. In each run, 50,000 operations were done, where an operation con-

Size	Ops/Sec	% Slower
None	417	0%
64	366	12.2%
128	348	16.5%
256	329	21.1%
512	311	25.4%
1024	277	33.5%
8192	115	72.4%

[0271] Precheck Domain Sizes

[0272] Before presenting general results, we discuss a tradeoff in the implementation of the Read Prechecking algorithm. The Read Prechecking algorithm verifies each read by computing the codeword of the regions which the read intersects. Since one codeword is stored for each protection domain, the size of the region leads to a time-space tradeoff for this scheme. We present the performance of Read Prechecking with Data Codeword maintenance for a variety of sizes of protection domains from 64 bytes to the 8K page size of our machine. With small size protection domains, this scheme performs well, but may add 3%-6% to the space usage of the database. The scheme breaks even with hardware protection at about 1K protection domains. These results are shown in the above precheck Table.

[0273] Results

[0274] The table below provides our results in terms of cost of corruption protection for the various discussed herein:

Algorithm	Physical Corruption		Logical Corruption	Ops/Sec	% Slower
	Direct	Indirect	Indirect		
Baseline	None	None	None	417	0%
Data CodeWord (CW)	Correct	None	Nothing	380	8.5%
Data CW w/Precheck, 64 byte	Correct	Prevent	Nothing	366	12.2%
Data CW w/ReadLog	Correct	Correct	Correct	345	17.1%
Data CW w/CW ReadLog	Correct	Correct	Correct	323	22.4%
Data CW w/Precheck, 512 byte	Correct	Prevent	Nothing	311	25.4%
Memory Protection	Prevent	Unneeded	Nothing	257	38.2%
Data CW w/Precheck, 8K byte	Correct	Prevent	Nothing	115	72.4%

sists of updating the (non-key) balance fields of one account, teller and branch, and adding a record to the history table. Transactions were committed after 500 operations, so that commit times do not dominate. The alternative was to design a highly concurrent test with group commits, introducing a great deal of complexity and variability into the test. Each test was run six times, and the results averaged. The results are reported in terms of number of operations completed per second.

[0269] Prechecking and Protection Domain Size

[0270] The table below provides operations per second and percent slower for page sizes from 64 bytes to 8 k bytes.

[0275] In the above Cost of Protection Table, a representative selection of the algorithms discussed in this paper are shown, along with the average number of operations per second the algorithm achieved in our tests, and the relative slowdown of the algorithm compared to the baseline algorithm, which is just the system running with no corruption protection. Our experiments show that detection of direct corruption can be achieved very cheaply, with about 8% cost, with simple data codeword protection. The choice between these algorithms can be made on ease of implementation in a particular system. Prechecking with a small domain size is economical at a 12% cost, depending on the acceptability of a 6% space overhead. Read logging lowers the space overhead, but raises the cost to 17%, which is

significant, but may be worthwhile, since automatic support for repairing the database can then be employed, and the results of erroneous transactions can be tracked. Logging the checksum of the data read, which increases the accuracy of the corruption recovery algorithms, adds 5% to the cost, bringing it to 22%. Memory protection using the standard mprotect call costs 38%, more than double the performance hit of codeword protection with read logging. Finally, pre-checking with large domain sizes fares very poorly.

[0276] Our conclusion from these results is that some form of codeword protection should be implemented in any DBMS in which application code has direct access to database data. Detection of direct corruption is quite cheap, and as limited as it is, is still far better than allowing corruption to remain undetected in the database. Other levels of protection may be implemented or offered to users so that they may make their own safety/performance tradeoff.

[0277] Thus there has been described a variety of schemes for preventing or detecting physical corruption using codewords, and for tracing and recovering from physical and logical corruption using read logging. Finally, a performance study comparing alternative techniques of corruption detection and recovery demonstrated the utility and practicality of the present invention involving codewording and read logging. Protection from direct physical corruption is economical to implement, transaction-carried corruption can be prevented cheaply if enough space is available for small protection domains, and detection of transaction-carried corruption for later correction through read logging imposes about a 17% cost on update transaction performance. The technique of the present invention opens up interesting possibilities in tracing logical errors through the database system and aiding in their correction. The new schemes are shown to be significantly cheaper than using the memory protection and unprotection provided by UNIX around every update.

[0278] Our techniques may deal with logical corruption and handle errors which are not caught by integrity constraints. Our techniques for handling physical corruption will be of increasing importance since applications are increasingly being provided direct access to persistent data. Since limited protection is very cheap, we believe implementors of database systems in which application code has direct access to database buffers should provide some form of protection, at least as an option for users. Our techniques have been shown to be highly portable, and use only simple integer operations which will be efficient on all modern processors.

[0279] In summary, then, the present invention may find application in the efficient recovery from physical or logical corruption, off-line generation of consistent checkpoints to be used to check global integrity constraints, fault-induction tests of logical corruption recovery, and implementation of these techniques in other database management systems. The present invention may be implemented in a real-time environment, for example, a communications environment where the busy state of communications channels, circuits and the like are preserved in a central (at a switch or server) or distributed database management system. All patent applications and articles referenced herein should be deemed to be incorporated by reference as to their entire contents. The present invention should only be deemed to be limited in scope by the claims which follow.

What we claim is:

1. A method of detecting and recovering from data corruption of a database comprising the step of logging information about reads of a database to detect and recover from physical corruption of the data in the database, said physical corruption arising from bad writes of data to the database or corruption arising indirectly therefrom.

2. A method of detecting and recovering from data corruption as recited in claim 1 further comprising the step of:

storing data of the database in main volatile memory of a data processor.

3. A method of detecting and recovering from data corruption as recited in claim 1 further comprising the steps of

maintaining a log of writes of the database and combining said log of reads and said log of writes into a combined log of reads and writes.

4. A method of detecting and recovering from data corruption as recited in claim 1 further comprising the step of logging one of an identity of information read from and an identity of information written to the database.

5. A method of detecting and recovering from data corruption as recited in claim 1 further comprising the step of:

storing a codeword in a record of said read-logging information.

6. A method of detecting and recovering from data corruption as recited in claim 3 further comprising the step of storing a codeword corresponding to a value of a data item in a record of said combined log of reads and writes.

7. A method of detecting and recovering from data corruption as recited in claim 1 further comprising the step of:

protecting data of the database with codewords, one codeword for each region of the database.

8. A method as recited in claim 1 wherein recovery in a delete transaction model comprises deleting effects of transactions from a database image.

9. A method of detecting and recovering from data corruption as recited in claim 3, said recovery from data corruption comprising first and second phases, a first redo phase followed by an undo phase.

10. A method as recited in claim 9 wherein said redo phase comprises a forward scan of a log of read and write operations.

11. A method as recited in claim 10 wherein said forward scan adds an identity of data to a corrupt data table whenever said data is written by a corrupt transaction.

12. A method as recited in claim 10 wherein said forward scan adds an identity of data to a corrupt data table whenever said data has failed an audit.

13. A method as recited in claim 10 further comprising the step of maintaining a corrupt transaction table.

14. A method as recited in claim 13 wherein said forward scan adds transactions to said corrupt transaction table that are known corrupt transactions.

15. A method as recited in claim 13 wherein said forward scan adds transactions to said corrupt transaction table whenever data is read that is identified in the corrupt data table.

16. A method as recited in claim 13 further comprising the step of storing a codeword corresponding to a value of a data item in a record of said combined log of reads and writes wherein said forward scan adds transactions to said corrupt transaction table whenever said codeword does not match the current value of said data item read from a database.

17. A method as recited in claim 15 wherein in a redo phase, actions such as writes are applied to a database image unless a transaction is listed in the corrupt transaction table.

18. A method as recited in claim 10 wherein said undo phase comprises undoing portions of corrupt transactions.

19. A method as recited in claim 10 wherein said undo phase comprises undoing in progress transactions at the time of a data processor failure.

20. A method as recited in claim 1 for further recovering from logical corruption, the method comprising the step of maintaining a logical redo log.

21. A method as recited in claim 20 further comprising the additional steps of maintaining a transaction code for redoing transactions and of storing user inputs for a transaction in said logical redo log.

22. A method as recited in claim 21 further comprising the step of maintaining a commit record at a transaction level.

23. A method as recited in claim 20 comprising the further step of returning a database to a transaction consistent state prior to a first detected instance of possible data corruption.

24. A method as recited in claim 23 comprising the further step of rerunning transactions affected by the first detected instance of a possible data corruption in the same order as an original set of transactions.

25. A method as recited in claim 23 comprising the further step of deleting effects of transactions when logical information is unavailable to permit redoing transactions.

26. A method as recited in claim 1, wherein direct physical corruption has occurred to data in memory used for database cache, further comprising the step of removing corruption from cache pages without reflecting any corrupt data values in log records.

27. A method of detecting and recovering from data corruption as recited in claim 3 further comprising the step of responsive to the identity of a known corrupt data item, determining a probable source of the corruption from said combined log of writes and reads.

28. A method as recited in claim 27 wherein said determining step comprises the substep of processing said log of writes and reads backwards from an end of said combined log of writes and reads.

29. A method as recited in claim 28 further comprising the substeps of maintaining a table of suspect corrupt data items and, for each data item of said table, maintaining a set of known corrupt data items whose corruption would be explained if said data item were corrupt.

30. A method as recited in claim 29 further comprising the substep of maintaining a table of suspect corrupt transactions and, for each transaction of said table, maintaining said set of known corrupt data items whose corruption would be explained if said transaction were corrupt.

31. A method as recited in claim 30 further comprising the substeps of adding a transaction to the table of suspect corrupt transactions when a write of a data item in the suspect data table by said transaction is encountered in said combined log during said backwards processing step.

32. A method as recited in claim 31 further comprising the substep of adding the set of known corrupt data items associated with said data item in the suspect data table to the set of known corrupt data items associated with said transaction.

33. A method as recited in claim 30 further comprising the substeps of adding a data item to the suspect data table when a read of said data item by a transaction in the suspect transaction table is encountered in said combined log during said backwards processing step.

34. A method as recited in claim 33 further comprising the substep of adding the set of known corrupt data items associated with said transaction to the set of known corrupt data items associated with said data item.

35. A method as recited in claim 34 further comprising the step of stopping said backwards processing whenever a suspect transaction exists in the suspect transaction table such that the set of data items associated with said transaction contains all known corrupt data items, said transaction being output to said user as a possible source of said corruption.

36. A method as recited in claim 35 further comprising the steps of performing forward recovery from said transaction determined by said user as a source of said corruption.

37. A method as recited in claim 1 wherein said information about reads comprises one of a start and an end point and length of data read.

38. A method as recited in claim 1 wherein, for detecting logical corruption of data, said method comprises the step of logging a codeword of a logical state found.

39. A method as recited in claim 1, further comprising the step of logging lock information in a logical read log.

40. A method as recited in claim 39 wherein said lock information comprises a name of a data item and a type of lock.

41. A method as recited in claim 1 further comprising the step of ensuring a disc image of said database is free of corruption.

42. A method as recited in claim 1, wherein, in the event of logical corruption, said method comprises the step of logging the identity and codeword for a logical structure protected by a lock.

43. A method of detecting and recovering from data corruption of a database comprising the steps of logging information about reads of a database to detect and recover from logical corruption of the data in the database, maintaining a logical redo log and storing user inputs for a transaction in said logical redo log.

44. A data corruption recovery method as recited in claim 43 further comprising the step of performing forward recovery in said logical redo log from a determined point of initial corruption.

45. A corruption recovery method as recited in claim 44 further comprising the steps of saving log records until a commit or an abort for a transaction is seen and, if a commit is seen, then scanning the read log records to determine if a transaction has read corrupted data and marking the transaction as corrupt.

46. A corruption recovery method as recited in claim 45 further comprising the steps of, if a commit is seen and the transaction is marked corrupt, reexecuting the transaction logically and, responsive thereto, replacing logical redo records with redo records generated by the logical reexecution of the transaction and adding data written by the transaction to a corrupt data table.

47. A corruption recovery method as recited in claim 45 further comprising the steps of, if a commit is seen and the transaction is not marked corrupt, executing its logical redo records.

48. A corruption recovery method as recited in claim 44 further comprising the steps of saving log records until a commit or an abort for a transaction is seen and, if an abort is seen, discarding the log records.

49. A method of detecting and recovering from data corruption of a database comprising the steps of logging information about reads of a database to detect and recover from logical corruption of the data in the database and deleting the effects of transactions from an image of the database.

50. The method of detecting and recovering from data corruption as recited in claim 49 further comprising the step of logging lock information in said logical redo log.

51. A method of detecting and recovering from data corruption as recited in claim 49, said recovery from data corruption comprising first and second phases, a first redo phase followed by an undo phase.

52. A method as recited in claim 51 wherein said redo phase comprises a forward scan of a log of read and write operations.

53. A method as recited in claim 52 wherein said forward scan adds an identity of data to a corrupt data table whenever said data is written by a corrupt transaction.

54. A method as recited in claim 52 wherein said forward scan adds transactions to a corrupt transaction table whenever data is read that is identified in the corrupt data table.

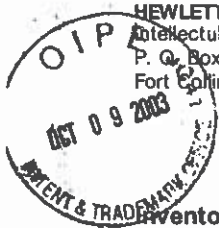
55. A method as recited in claim 54 wherein in a redo phase, actions such as writes are applied to a database image unless a transaction is listed in the corrupt transaction table.

56. A method as recited in claim 51 wherein said undo phase comprises undoing portions of corrupt transactions.

57. A method as recited in claim 51 wherein said undo phase comprises undoing in progress transactions at the time of a data processor failure.

58. A method of detecting and recovering from data corruption of a database comprising the steps of logging information about reads of a database to detect and recover from logical corruption of the data in the database and logging a codeword of a logical state found.

* * * * *



HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P. O. Box 272400
Fort Collins, Colorado 80527-2400

PATENT APPLICATION

ATTORNEY DOCKET NO. 10006924-1

IN THE
UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s): Ganesh, et al.

Confirmation No.: 6266

Application No.: 09/774,839

Examiner: Kindred, Alford W

Filing Date: 01/31/2001

Group Art Unit: 2172

Title: RECREATION OF ARCHIVES AT A DISASTER RECOVERY SITE

Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

RECEIVED

OCT 21 2003

Technology Center 2100

TRANSMITTAL LETTER FOR RESPONSE/AMENDMENT

Sir:

Transmitted herewith is/are the following in the above-identified application:

- (X) Response/Amendment () Petition to extend time to respond
() New fee as calculated below () Supplemental Declaration
(X) No additional fee (Address envelope to "Mail Stop Non-Fee Amendment")
() Other: (fee \$)

CLAIMS AS AMENDED BY OTHER THAN A SMALL ENTITY						
(1) FOR	(2) CLAIMS REMAINING AFTER AMENDMENT	(3) NUMBER EXTRA	(4) HIGHEST NUMBER PREVIOUSLY PAID FOR	(5) PRESENT EXTRA	(6) RATE	(7) ADDITIONAL FEES
TOTAL CLAIMS		MINUS		= 0	X \$18	\$ 0
INDEP. CLAIMS		MINUS		= 0	X \$86	\$ 0
[] FIRST PRESENTATION OF A MULTIPLE DEPENDENT CLAIM					+ \$290	\$ 0
EXTENSION FEE	1ST MONTH \$110.00	2ND MONTH \$420.00	3RD MONTH \$950.00	4TH MONTH \$1480.00		\$ 0
OTHER FEES						\$
TOTAL ADDITIONAL FEE FOR THIS AMENDMENT						\$ 0

Charge \$ 0 to Deposit Account 08-2025. At any time during the pendency of this application, please charge any fees required or credit any overpayment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16 through 1.21 inclusive, and any other sections in Title 37 of the Code of Federal Regulations that may regulate fees. A duplicate copy of this sheet is enclosed.

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Alexandria, VA 22313-1450.

Date of Deposit: October 6, 2003

Typed Name: Steve M. Perry

Signature: Steve M. Perry

Respectfully submitted,

Ganesh, et al.

By Steve M. Perry

Steve M. Perry

Attorney/Agent for Applicant(s)

Reg. No. 45,357

Date: October 6, 2003

Rev 10/03 (TransAmd)

Telephone No.: (801)566-6633

- Attach as First Page to Transmitted Papers -



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICANT: Ganesh, et al.

SERIAL NO.: 09/774,839

FILED: January 31, 2001

FOR: RECREATION OF ARCHIVES
AT A DISASTER RECOVERY
SITE

ART UNIT: 2172

EXAMINER: Alford W. Kindred

DOCKET: PDNO 10006924-1

AMENDMENT

RECEIVED
OCT 21 2003
Technology Center 2100

Mail Stop Non-Fee Amendment
Commissioner for Patents
P. O. Box 1450
Alexandria, VA 22313-1450

OK to enter
Amend

Sir:

In response to the Office Action mailed August 5, 2003, Applicant offers the following Amendment and request reconsideration of the above-captioned application. This Amendment addresses every ground of rejection set forth in the Office Action.

CERTIFICATE OF DEPOSIT

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postage prepaid, in an envelope addressed to: Box Non-Fee Amendment, Commissioner for Patents, Washington, D.C. 20231, on October 6, 2003.

Respectfully submitted,


Steve M. Perry, Reg. 45,357

B

In the Claims:

B/ 1. (Previously Amended) A method for creating a standby data processing system located at a standby site, which is remotely located from a primary site, comprising the steps of:

(a) computing a difference between an active transaction redo log and an archive transaction redo log to produce a delta image at the primary site;

(b) transmitting the active transaction redo log and the delta image, separately in time sequence, from the primary site to the standby site; and

(c) combining the delta image with the active transaction redo log at the standby site, to produce a standby archive transaction redo log.

2. (Cancelled.)

3. (Cancelled.)

2
A. (Previously Amended) A method as in claim 1, further comprising the step of storing the standby archive transaction redo log on the standby site.

3
B. (Previously Amended) A method as in claim 1, further comprising the step of storing transactions in the standby active transaction redo log as they are trapped at the primary site.

B1 ~~4~~ (Previously Amended) A method as in claim 1, further comprising the step of entering transactions from the archive transaction redo log into the standby transaction processing system.

~~5~~ (Currently Amended) A method for creating archive redo logs for a standby database at a remote location, comprising the steps of:

- (a) trapping active log writes using a remote mirroring device that is capable of mirroring the log writes to a standby site, to create an online redo log at the standby site;
- (b) computing a difference between an online redo log and archive redo log to create a delta image at a primary database, wherein the delta image includes an archive header;
- (c) transmitting the delta image to a standby archive log replicator contained in the standby database; and
- (d) combining the delta image and archive header with the online redo log to create a standby archive redo log on the standby database.

~~6~~ ⁵ A method as in claim ~~7~~, further comprising the step of applying the standby archive redo log by creating links to the standby archive redo log.

~~7~~ ⁵ A method as in claim ~~7~~, further comprising the step of transmitting the delta image to a standby archive log replicator using a first remote mirroring device in the primary database and a second remote mirroring device in the standby database.

B1
~~8~~ 10. A method as in claim ~~9~~ ⁷, further comprising the step of using replication handles to track the progress of mirror writes on the standby database as compared to the primary database.

~~9~~ 11. A method as in claim ~~10~~ ⁸, further comprising the step of checking the primary database queries using the archive log replicator to determine whether archiving is complete.

~~10~~ 12. A method as in claim ~~7~~ ⁵, further comprising the step of shipping an entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously after archiving becomes slow enough that there is a risk of primary database blocking.

~~11~~ 13. A method as in claim ~~12~~ ¹⁰, further comprising the step of shipping an entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously when a spool buffer in a remote mirroring system has been filled.

~~12~~ 14. A method for creating archive redo logs from a primary database, for a standby database at a remote location, comprising the steps of:

- (a) trapping file system transactions to create an online redo log at the standby database;
- (b) computing a difference between an online redo log and archive redo log in the primary database to find the delta image of the archive redo log;
- (c) transmitting the delta image to a standby database; and
- (d) combining the delta image with the online redo log in the standby database to create standby archive redo log on the standby database.

13
~~13.~~ A method as in claim ~~14~~¹², further comprising the step of applying the standby archive redo log by creating a link between the standby database and the standby archive redo log.

14
~~14.~~ A method as in claim ~~14~~¹², further comprising the step of transmitting the delta image to a standby archive log replicator using a first remote mirroring device in the primary database and a second remote mirroring device in the standby database.

15
~~15.~~ A method as in claim ~~14~~¹⁴, further comprising the step of using replication handles to track the progress of mirror writes on the standby database as compared to the primary database.

16
~~16.~~ A method as in claim ~~14~~¹², further comprising the step of checking the primary database with database queries using the archive log replicator to determine whether archiving is complete.

17
~~17.~~ A method as in claim ~~14~~¹², further comprising the step of shipping an entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously after archiving becomes slow enough that there is a risk of the primary database blocking.

18
~~18.~~ A method as in claim ~~14~~¹², further comprising the step of shipping the entire archive redo log from the primary archive log replicator to the standby archive log replicator asynchronously when a spool buffer in a remote mirroring system has been filled.

B1

¹⁹
~~21.~~ A system for creating a standby data processing system located at a standby site, which is remotely located from a primary site, comprising the steps of:

- means for computing a difference between an active data file and an archive data file to produce a delta image at the primary site;
- means for transmitting the active data file and the delta image, separately in time sequence, from the primary site to the standby site; and
- means for combining the delta image with the active data file at the standby site, to produce a standby archive data file.

²⁰
~~22.~~ An article of manufacture comprising:

- a computer usage medium having computer readable program code means embodied therein for creating archive redo logs for a standby database at a remote location, the computer readable program code means in said article of manufacture comprising:
 - computer readable program code means for trapping active log writes using a remote mirroring device that is capable of mirroring the log writes to a standby site, to create an online redo log at the standby site;
 - computer readable program code means for computing a difference between an online redo log and archive redo log to create a delta image at a primary database;
 - computer readable program code means for transmitting the delta image to a standby archive log replicator contained in the standby database; and
 - computer readable program code means for combining the delta image with the online redo log to create a standby archive redo log on the standby database.

Remarks

Applicant expresses appreciation to the Examiner for the consideration of the subject patent application. The Office Action rejected claims 1, 4, and 6-22 under 35 U.S.C. 103(c), as being unpatentable over Tamer, et al. (U.S. No. 6,035,412) in view of Bohannon, et al. (U.S. Patent Publication 2002/0091718).

The Office Action has maintained the position that the language in claim 1 of “transmitting a delta image for an archive transaction redo log to a standby site ...” is taught by Tamer who synchs portions of a backup database in relation to a primary database. The Examiner stated that synching a backup redo log to a primary redo log is the same as computing the difference or delta image between the active transaction redo log and the archive redo log.

Applicant would like to explain why the correlation made by the Office Action is not appropriate for the present invention. One significant difference between the present invention and the prior art is that Tamer and Bohannon process or sync the **log data** in the redo logs themselves, but the present invention is using the **database system information** that makes a log function as either an active log or an archive log.

A redo log file contains all the data needed to perform database recovery and the redo log records every change made to the database structure. (Specification, page 8, paragraph 2.) Further, the active transaction redo log and the archive transaction redo log contain duplicate copies of this recovery information. When the active redo log and archive redo log are synchronized properly by the “dirty track” method, then the redo logs contained in the active and archive redo log files will be the same.

If the active redo log and archive redo log contain the same recovery logs when properly synched, then how does the database system tell the difference between these two files? Specifically, the database system has an archive header and other database system identification data that marks the archive transaction redo logs.

Claim 1 reflects these differences by including the operation of “computing a difference between an active transaction redo log and an archive transaction redo log to

produce a delta image at the primary site.” This language claims the delta image used to create an archive transaction redo log, which cannot be done with the prior art references.

Claim 7 has also been amended to include the operation of “computing a difference between an online redo log and archive redo log to create a delta image at a primary database, **wherein the delta image includes an archive header.**” This further identifies and points out the distinct features of the delta image as discussed.

In other words, the delta file does not include the redo logs or dirty track information. This is because the redo log portions of the active and archived redo logs are identical at the standby site. The active transaction redo log is shipped over the WAN with the delta image, which includes the file header information. The delta image is then overlaid on the active transaction redo log to create the archive transaction redo log. In effect, this avoids shipping the redo log information twice.

Detailed Discussion of Tamer under 35 U.S.C. 103(c)

FIG. 1A of Tamer illustrates that Tamer is only interested in the interaction of the local mirroring between the two data storage systems 2 and 4. These two data storage systems in Tamer loosely correspond to 50 and 52 in FIG. 3 of the present invention.

Both the present invention and Tamer include bit maps or what are known as “dirty track maps” but these bit maps are applied to the local mirroring as illustrated in FIG. 1A of Tamer where mirroring occurs between the first local data storage system 2 and the second local data storage system 4. (Column 7, lines 41-67) In one embodiment, the present invention uses bit maps in its remote mirroring system 66, 68 and online logs 58, but the bit maps are separate and distinct from the delta image. As discussed above, Tamer does not teach or suggest enabling the creation of an archive transaction redo log on a standby site by “combining the delta image with the active transaction redo log at the standby site” as claimed.

Tamer does not teach or suggest transmitting a delta image for an archive transaction redo log to a standby site as claimed in the present invention. Specifically, claim 1 includes

the patentably distinct language of “transmitting the active transaction redo log and the delta image, separately in time sequence, from the primary site to the standby site.”

Discussion of Bohannon under 35 U.S.C. 103(c)

The Office Action has also rejected the claims over Tamer in view of Bohannon, et al. Bohannon will now be discussed in relation to the present invention. The Office Action stated that Bohannon teaches or suggests “combining the delta image with the active transaction redo log at the standby site, to produce a standby archive” (Page 3, paragraph [0030] and page 8, paragraph [0099].)

Bohannon actually teaches a system for preventing the corruption of data in a database using codewords for word-wise parity checking. This is described in paragraphs [0092] through [0100] of Bohannon. Particularly, paragraph [0093] of Bohannon states that “...codewords are maintained for page-sized or smaller regions of data known as protection regions.” Further, Bohannon teaches, “By using word-wise parity, or a similar scheme, updates to the codeword can be made based only on the portion of the protection region which was updated...” Paragraph [0094] reiterates that the codewords are being used for corruption detection and not for a delta image or standby site.

Paragraph [0099] of Bohannon was cited by the Office Action as teaching the combining of the delta image with the active transaction redo log to produce a standby archive. A detailed reading of paragraphs [0099] and [0100] reveals that Bohannon teaches using an exclusive OR to combine the codeword bits and the changes in the bits between the undo image U and the redo image R. The ORed information is used to generate a new codeword. This codeword (or parity word) is different than the present invention because the present invention does not process the redo information in the file, but creates a delta image to capture the database system information (not redo log information) in the archive transaction redo log.

Even if Tamer and Bohannon were combined, the resulting combination would only provide a database that uses a dirty track bitmap to identify portions of the backup database that are out-of-sync with the primary database (Col. 7, lines 40-64) and parity checking

codewords in the database. The combination of Tamer and Bohannon does not teach or suggest how the data transfer to a standby site can be reduced by using a delta image to create an archive transaction redo log at a standby site. Particularly, the combination of Tamer and Bohannon do not teach or suggest, “combining the delta image with the active transaction redo log at the standby site, to produce a standby archive transaction redo log.”

With regard to claim 5, Tamer and Bohannon do not teach “storing transactions in the standby active transaction redo log as they are trapped at the primary site.” Specifically, Tamer and Bohannon do not teach or suggest any operations with respect to a standby site. Particularly, Tamer describes two local data storage systems that are connected by a high speed link 6. (Col. 7 lines 33-36.) FIG. 1 of Bohannon also illustrates a local database system. In contrast, the present invention focuses on reducing the amount of data transfer to a standby cluster 53 over a lower bandwidth network as illustrated in FIG. 3 by the WAN 73. The master and slave, described in column 4, lines 32-49 of Tamer, refer only to a data storage system located at a primary site and not at a standby site. Claim 6 includes further patentably distinct subject matter when considered in light of claim 1 from which claim 6 depends.

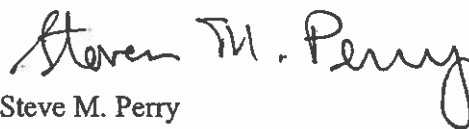
Claims 7-22 were also rejected by the Office Action under the same arguments applied to claims 1-6. The method claims of present invention also include the steps of combining the online transaction log 80 with the delta image to form the archive transaction log(s) 76, 78 at the standby site, and this is not taught or suggested by the prior art. The same arguments that have been presented by the Applicant also apply to these claims 7-22 and claims 1-22 should be allowed.

If any impediment to the allowance of these claims remains after consideration of this Amendment, the Examiner is invited to call Steve Perry at (801) 566-6633 so that any necessary changes to the claims can be considered.

The Commissioner is hereby authorized to charge any additional fee or to credit any overpayment in connection with this Amendment to Deposit Account No. 08-2025.

DATED this 6th day of October, 2003.

Respectfully submitted,

A handwritten signature in black ink that reads "Steven M. Perry". The signature is written in a cursive style with a large, looped "P" at the end.

Steve M. Perry
Attorney for Applicant
Registration No. 45,357

THORPE, NORTH & WESTERN, L.L.P.
P.O. Box 1219
Sandy, Utah 84091-1219
Telephone (801) 566-6633

Notice of Allowability	Application No.	Applicant(s)	
	09/774,839	GANESH ET AL.	
	Examiner	Art Unit	
	Alford W. Kindred	2172	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to Amendment B, filed 10/9/03.

2. ☒ The allowed claim(s) is/are 1 and 4-22, renumbered as 1-20.

3. ☒ The drawings filed on 8/13/01 are accepted by the Examiner.

4. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a) ☐ All b) ☐ Some* c) ☐ None of the:

1. ☐ Certified copies of the priority documents have been received.

2. ☐ Certified copies of the priority documents have been received in Application No. _____.

3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

* Certified copies not received: _____.

5. ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application).

(a) ☐ The translation of the foreign language provisional application has been received.

6. ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application. **THIS THREE-MONTH PERIOD IS NOT EXTENDABLE**

7. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.

8. ☐ CORRECTED DRAWINGS must be submitted.

(a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached

1) ☐ hereto or 2) ☐ to Paper No. _____.

(b) ☐ including changes required by the proposed drawing correction filed _____, which has been approved by the Examiner.

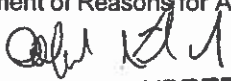
(c) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No. _____.

Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet.

9. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1 ☐ Notice of References Cited (PTO-892)	2 ☐ Notice of Informal Patent Application (PTO-152)
3 ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	4 ☐ Interview Summary (PTO-413), Paper No. _____.
5 ☒ Information Disclosure Statements (PTO-1449), Paper No. <u>8</u> .	6 ☐ Examiner's Amendment/Comment
7 ☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material	8 ☒ Examiner's Statement of Reasons for Allowance
	9 ☐ Other


ALFORD KINDRED
PRIMARY EXAMINER

DETAILED ACTION

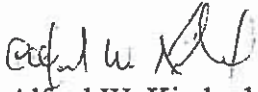
Allowable Subject Matter

1. Claims 1 and 4-22, renumbered as 1-20, are allowed over the prior art of record.
2. The following is an examiner's statement of reasons for allowance:

Regarding Applicant's claimed invention of "computing a difference between an active transaction redo log and an archive transaction redo log to produce a delta image . . . transmitting the active transaction redo log and the delta image, separately in time . . .", combined with "combining the delta image with the active transaction redo log at the standby site, to produce a standby archive transaction redo log."

Conclusion

3. Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."
4. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Alford W. Kindred whose telephone number is 703-305-3802. The examiner can normally be reached on Mon-Friday, 8:00am-5:00pm.
5. If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kim Vu can be reached on 703-305-4393. The fax phone numbers for the organization where this application or proceeding is assigned are 703-746-7239 for regular communications and 703-746-7238 for After Final communications.
6. Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is 703-305-3900.


Alford W. Kindred
Patent Examiner
Art Unit# 2172
October 31, 2003



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

NOTICE OF ALLOWANCE AND FEE(S) DUE

7590 11/04/2003
HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, CO 80527-2400

EXAMINER	
KINDRED, ALFORD W	
ART UNIT	PAPER NUMBER
2172	
DATE MAILED: 11/04/2003	

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/774,839	01/31/2001	Shivaji Ganesh	10006924-1	6266

TITLE OF INVENTION: RECREATION OF ARCHIVES AT A DISASTER RECOVERY SITE

APPLN. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	\$1330	\$300	\$1630	02/04/2004

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. **PROSECUTION ON THE MERITS IS CLOSED.** THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN **THREE MONTHS** FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. **THIS STATUTORY PERIOD CANNOT BE EXTENDED.** SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE REFLECTS A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE APPLIED IN THIS APPLICATION. THE PTOL-85B (OR AN EQUIVALENT) MUST BE RETURNED WITHIN THIS PERIOD EVEN IF NO FEE IS DUE OR THE APPLICATION WILL BE REGARDED AS ABANDONED.

HOW TO REPLY TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

- A. If the status is the same, pay the TOTAL FEE(S) DUE shown above.
- B. If the status is changed, pay the PUBLICATION FEE (if required) and twice the amount of the ISSUE FEE shown above and notify the United States Patent and Trademark Office of the change in status, or

If the SMALL ENTITY is shown as NO:

- A. Pay TOTAL FEE(S) DUE shown above, or
- B. If applicant claimed SMALL ENTITY status before, or is now claiming SMALL ENTITY status, check the box below and enclose the PUBLICATION FEE and 1/2 the ISSUE FEE shown above.
- ☐ Applicant claims SMALL ENTITY status.
See 37 CFR 1.27.

II. PART B - FEE(S) TRANSMITTAL should be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). Even if the fee(s) have already been paid, Part B - Fee(s) Transmittal should be completed and returned. If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Mail Stop ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: **Mail** Mail Stop ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or **Fax** (703) 746-4000

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 4 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Legibly mark-up with any corrections or use Block 1)

7590 11/04/2003
HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, CO 80527-2400

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission
I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO, on the date indicated below.

(Depositor's name)
(Signature)
(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/774,839	01/31/2001	Shivaji Ganesh	10006924-1	6266

TITLE OF INVENTION: RECREATION OF ARCHIVES AT A DISASTER RECOVERY SITE

APPLN. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	NO	\$1330	\$300	\$1630	02/04/2004

EXAMINER	ART UNIT	CLASS-SUBCLASS
KINDRED, ALFORD W	2172	707-204000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363). ☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached. ☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. Use of a Customer Number is required.	2. For printing on the patent front page, list (1) the names of up to 3 registered patent attorneys or agents OR, alternatively, (2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed. 1 _____ 2 _____ 3 _____
---	--

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)
PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. Inclusion of assignee data is only appropriate when an assignment has been previously submitted to the USPTO or is being submitted under separate cover. Completion of this form is NOT a substitute for filing an assignment.
(A) NAME OF ASSIGNEE (B) RESIDENCE: (CITY AND STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent); ☐ individual ☐ corporation or other private group entity ☐ government

4a. The following fee(s) are enclosed: ☐ Issue Fee ☐ Publication Fee ☐ Advance Order - # of Copies _____	4b. Payment of Fee(s): ☐ A check in the amount of the fee(s) is enclosed. ☐ Payment by credit card. Form PTO-2038 is attached. ☐ The Director is hereby authorized by charge the required fee(s), or credit any overpayment, to Deposit Account Number _____ (enclose an extra copy of this form).
--	--

Director for Patents is requested to apply the Issue Fee and Publication Fee (if any) or to re-apply any previously paid issue fee to the application identified above.

(Authorized Signature)	(Date)
NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office. This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Alexandria, Virginia 22313-1450. Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.	

TRANSMIT THIS FORM WITH FEE(S)



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/774,839	01/31/2001	Shivaji Ganesh	10006924-1	6266
7590 11/04/2003				
HEWLETT-PACKARD COMPANY Intellectual Property Administration P.O. Box 272400 Fort Collins, CO 80527-2400			EXAMINER KINDRED, ALFORD W	
			ART UNIT 2172	PAPER NUMBER

DATE MAILED: 11/04/2003

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment to date is 303 day(s). If the issue fee is paid on the date that is three months after the mailing date of this notice and the patent issues on the Tuesday before the date that is 28 weeks (six and a half months) after the mailing date of this notice, the Patent Term Adjustment will be 303 day(s).

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) system (<http://pair.uspto.gov>).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (703) 305-1383. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at (703) 305-8283.



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/774,839	01/31/2001	Shivaji Ganesh	10006924-1	6266
7590 11/04/2003				
HEWLETT-PACKARD COMPANY Intellectual Property Administration P.O. Box 272400 Fort Collins, CO 80527-2400			EXAMINER KINDRED, ALFORD W	
			ART UNIT 2172	PAPER NUMBER
DATE MAILED: 11/04/2003				

Notice of Fee Increase on October 1, 2003

If a reply to a "Notice of Allowance and Fee(s) Due" is filed in the Office on or after October 1, 2003, then the amount due will be higher than that set forth in the "Notice of Allowance and Fee(s) Due" since there will be an increase in fees effective on October 1, 2003. See Revision of Patent Fees for Fiscal Year 2004; Final Rule, 68 Fed. Reg. 41532, 41533, 41534 (July 14, 2003).

The current fee schedule is accessible from (<http://www.uspto.gov/main/howtofees.htm>).

If the fee paid is the amount shown on the "Notice of Allowance and Fee(s) Due" but not the correct amount in view of the fee increase, a "Notice of Pay Balance of Issue Fee" will be mailed to applicant. In order to avoid processing delays associated with mailing of a "Notice of Pay Balance of Issue Fee," if the response to the Notice of Allowance is to be filed on or after October 1, 2003 (or mailed with a certificate of mailing on or after October 1, 2003), the issue fee paid should be the fee that is required at the time the fee is paid. If the issue fee was previously paid, and the response to the "Notice of Allowance and Fee(s) Due" includes a request to apply a previously-paid issue fee to the issue fee now due, then the difference between the issue fee amount at the time the response is filed and the previously-paid issue fee should be paid. See Manual of Patent Examining Procedure, Section 1308.01 (Eighth Edition, August 2001).

Effective October 1, 2003, 37 CFR 1.18 is amended by revising paragraphs (a) through (c) to read as set forth below.

Section 1.18 Patent post allowance (including issue) fees.

- (a) Issue fee for issuing each original or reissue patent, except a design or plant patent:
- | | |
|---------------------------------------|------------|
| By a small entity (Sec. 1.27(a))..... | \$665.00 |
| By other than a small entity..... | \$1,330.00 |
- (b) Issue fee for issuing a design patent:
- | | |
|---------------------------------------|----------|
| By a small entity (Sec. 1.27(a))..... | \$240.00 |
| By other than a small entity..... | \$480.00 |
- (c) Issue fee for issuing a plant patent:
- | | |
|---------------------------------------|----------|
| By a small entity (Sec. 1.27(a))..... | \$320.00 |
| By other than a small entity..... | \$640.00 |

Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at (703) 305-8283.

970 AB 7247 P.01

**PART B - FEE(S) TRANSMITTAL**

and send this form, together with applicable fee(s), to: Mail

Mail Stop ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
(703) 746-4000

or Fax (703) 746-4000

INSTRUCTIONS: This form should be used for transmitting the **ISSUE FEE** and **PUBLICATION FEE** (if required). Blocks 1 through 4 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address, and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

NOTE: A certificate of mailing can only be used for domestic mailings of the Patent. This certificate cannot be used for any other accompanying documents.

Note: A certificate of mailing can only be used for domestic mailings of the **Feet's Transmittal**. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmittal
I hereby certify that this Patent Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being otherwise submitted to the USPTO, on the date indicated below.

Submitted to the CSST (C) on the date indicated below
 Laura M. Clark
 Signature of subject
 Signature
 Date 12/4/2003

7900 11 04 2003
HEWLETT-PACKARD COMPANY
 Intellectual Property Administration
 P.O. Box 272400
 Fort Collins, CO 80527-2400

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09-374,339	01/31/2001	Shirley Ganeish	10006PC-1	1366

09-174,939 01/31/2001
TITLE OF INTENTION: RECREATION OF ARCHIVES AT A DISASTER RECOVERY SITE

APPLX. TYPE	SMALL ENTITY	ISSUE FEE	PUBLICATION FEE	TOTAL FEE: \$100	DAYS DUE
nonprovisional	NO	\$1330	\$300	\$1630	02/04/2004
EXAMINER		ART UNIT	CLASS-SUBCLASS		
KINDRED, ALFORD W		3172	107-20-0000		

1. Change of correspondence address or indication of "Free Address" (37 CFR 1.563)

☐ Change of correspondence address (or Change of Correspondence Address form PTO-59-131 attached).

☐ "Fee Address" indication (or "Fee Address" Indication form PTO-SB-7; Rev 03-02 or more recent) attached. Use of a Customer Number is required.

3. For printing on the patent front page, list (1) the names of up to 3 registered patent attorneys or agents (OR, alternatively, (2) the name of a single firm (acting as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents, if no name is listed, no name will be printed.

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (770141 07P)

ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. Inclusion of assignee data is only appropriate when an assignment has been previously submitted to the USPTO or is being submitted under separate cover. Completion of this form is NOT a substitute for filing of assignment.

(B) RESIDENCE: (CITY and STATE OR COUNTRY)

(A) NAME OF ASSIGNEE
Hoydell-Packard Development Company, L.P.

Houston, Texas

Please check the appropriate assignee category or categories (will not be printed on the patent): ☐ individual ☒ corporation or other private group entity ☐ governmental

4a. The following text(s) are enclosed:

Issue For

Publication Fee

☐ Advance Order - # of Copies

4th. Payment of Fees:

☐ A check in the amount of the fee(s) is enclosed.

☐ Payment by credit card. Form PTO-2025 is attached.

X The Director is hereby authorized to charge the required fees, or credit any overpayment, to Deposit Account Number 08-2025 (enclose an extra copy of this form).

Director for Patents is requested to apply the Issue Fee and Publication Fee (if any) or to re-apply any previously paid issue fee to this application.

(Authorized Signatory) T. A. R. (Date) 12-7-73

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignor or other party in interest, as shown by the records of the United States Patent and Trademark Office.

OTHER THAN THE INFORMATION CONTAINED HEREIN, THIS DOCUMENT CONTAINS NO INFORMATION OF A NATURE THAT IT IS UNLAWFUL FOR ANY PERSON TO DISCLOSE TO THE PUBLIC. THIS DOCUMENT IS NOT TO BE RELEASED TO THE PUBLIC IN ANY MANNER THAT WOULD REVEAL INFORMATION OF A NATURE THAT IT IS UNLAWFUL FOR ANY PERSON TO DISCLOSE TO THE PUBLIC. THIS DOCUMENT IS NOT TO BE RELEASED TO THE PUBLIC IN ANY MANNER THAT WOULD REVEAL INFORMATION OF A NATURE THAT IT IS UNLAWFUL FOR ANY PERSON TO DISCLOSE TO THE PUBLIC.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

TRANSMIT THIS FORCE WITH FEELS!

ONID 06.11-0003 U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

PAGE 1/2 * RCVD AT 12/4/2003 11:22:52 AM [Eastern Standard Time] * SVR:USPTO-EFXRF-2/1 * DNS:7464000 * CSID:970 898 7247 * DURATION (mm:ss):01:34

12/04/2003 3DIRETAE 00000151 082025 09774839

1330.00 DA
300.00 DA

File History Content Report

The following content is missing from the original file history record obtained from the United States Patent and Trademark Office. No additional information is available.

Document Date - 2004-02-10
Document Title - USPTO Grant

8412372



THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

October 10, 2023

THIS IS TO CERTIFY THAT ANNEXED IS A TRUE COPY FROM THE
RECORDS OF THIS OFFICE OF THE FILE WRAPPER AND CONTENTS
OF:

APPLICATION NUMBER: 09/774,839

FILING DATE: *January 31, 2001*

PATENT NUMBER: 6,691,139

ISSUE DATE: *February 10, 2004*

By Authority of the
Under Secretary of Commerce for Intellectual Property
and Director of the United States Patent and Trademark Office



W. Montgomery
Wanda Montgomery
Certifying Officer

PATENT APPLICATION FEE DETERMINATION RECORD
Effective October 1, 2000

10006924-1

CLAIMS AS FILED - PART I

	(Column 1)	(Column 2)
TOTAL CLAIMS	22	
FOR	NUMBER FILED	NUMBER EXTRA
TOTAL CHARGEABLE CLAIMS	22 minus 20 =	* 2
INDEPENDENT CLAIMS	5 minus 3 =	* 2
MULTIPLE DEPENDENT CLAIM PRESENT ☐		

SMALL ENTITY
TYPE ☐

OR
OTHER THAN
SMALL ENTITY

RATE	FEE
BASIC FEE	355.00
X\$ 9=	
X40=	
+135=	
TOTAL	

RATE	FEE
BASIC FEE	710.00
X\$18=	36
X80=	160
+270=	
TOTAL	906

* If the difference in column 1 is less than zero, enter "0" in column 2

CLAIMS AS AMENDED - PART II

	(Column 1)	(Column 2)	(Column 3)
AMENDMENT A	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
	Total *	Minus **	=
	Independent *	Minus ***	=
	FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM ☐		

SMALL ENTITY

OR
OTHER THAN
SMALL ENTITY

RATE	ADDITIONAL FEE
X\$ 9=	
X40=	
+135=	
TOTAL ADDIT. FEE	

RATE	ADDITIONAL FEE
X\$18=	
X80=	
+270=	
TOTAL ADDIT. FEE	

	(Column 1)	(Column 2)	(Column 3)
AMENDMENT B	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
	Total *	Minus **	=
	Independent *	Minus ***	=
	FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM ☐		

RATE	ADDITIONAL FEE
X\$ 9=	
X40=	
+135=	
TOTAL ADDIT. FEE	

RATE	ADDITIONAL FEE
X\$18=	
X80=	
+270=	
TOTAL ADDIT. FEE	

	(Column 1)	(Column 2)	(Column 3)
AMENDMENT C	CLAIMS REMAINING AFTER AMENDMENT	HIGHEST NUMBER PREVIOUSLY PAID FOR	PRESENT EXTRA
	Total *	Minus **	=
	Independent *	Minus ***	=
	FIRST PRESENTATION OF MULTIPLE DEPENDENT CLAIM ☐		

RATE	ADDITIONAL FEE
X\$ 9=	
X40=	
+135=	
TOTAL ADDIT. FEE	

RATE	ADDITIONAL FEE
X\$18=	
X80=	
+270=	
TOTAL ADDIT. FEE	

* If the entry in column 1 is less than the entry in column 2, write "0" in column 3.

** If the "Highest Number Previously Paid For" IN THIS SPACE is less than 20, enter "20."

***If the "Highest Number Previously Paid For" IN THIS SPACE is less than 3, enter "3."

The "Highest Number Previously Paid For" (Total or Independent) is the highest number found in the appropriate box in column 1.

CLAIMS ONLY

SERIAL NO.
09774839

FILING DATE
01-31-01

APPLICANT(S)
Steve M. Perry

CLAIMS						
	AS FILED		AFTER 1st AMENDMENT		AFTER 2nd AMENDMENT	
	IND.	DEP.	IND.	DEP.	IND.	DEP.
1	1					
2		1				
3		1				
4		1				
5		1				
6		1				
7	1					
8		1				
9		1				
10		1				
11		1				
12		1				
13		1				
14	1					
15		1				
16		1				
17		1				
18		1				
19		1				
20		1				
21	1					
22	1					
23						
24						
25						
26						
27						
28						
29						
30						
31						
32						
33						
34						
35						
36						
37						
38						
39						
40						
41						
42						
43						
44						
45						
46						
47						
48						
49						
50						
TOTAL IND.	5					
TOTAL DEP.	17					
TOTAL CLAIMS	22					

	*		*		*	
	IND.	DEP.	IND.	DEP.	IND.	DEP.
51						
52						
53						
54						
55						
56						
57						
58						
59						
60						
61						
62						
63						
64						
65						
66						
67						
68						
69						
70						
71						
72						
73						
74						
75						
76						
77						
78						
79						
80						
81						
82						
83						
84						
85						
86						
87						
88						
89						
90						
91						
92						
93						
94						
95						
96						
97						
98						
99						
100						
TOTAL IND.						
TOTAL DEP.						
TOTAL CLAIMS						

* MAY BE USED FOR ADDITIONAL CLAIMS OR ADMENDMENTS

Search results for: PNS=(US6691139);

Collections searched: DWPI; ARIPO grant; Argentina UM; Argentina app; Armenia grant; Australia UM; Australia app; Australia grant; Austria UM; Austria app; Austria grant; Belarus UM; Belarus grant; Belgium app; Belgium grant; Brazil UM; Brazil app; Brazil grant; Bulgaria UM; Bulgaria app; Bulgaria grant; Canada app; Canada grant; China, Mainland UM; China, Mainland app; China, Mainland grant; Colombia app; Costa Rica UM; Costa Rica app; Croatia UM; Croatia app; Croatia grant; Cuba app; Cuba grant; Czech Republic UM; Czech Republic app; Czech Republic grant; Czechoslovakia grant; Denmark UM; Denmark app; Denmark grant; Estonia UM; Estonia app; Estonia grant; Eurasian (EAPO) grant; Europe (EP) app; Europe (EP) grant; Finland app; Finland grant; France app; France grant; Georgia grant; German Democratic Republic app; German Democratic Republic grant; Germany UM; Germany app; Germany grant; Greece app; Greece grant; Gulf Cooperation Council grant; Hong Kong UM; Hong Kong app; Hong Kong grant; Hungary grant; Iceland grant; India app; India grant; Indonesia UM; Indonesia app; Ireland UM; Ireland app; Ireland grant; Israel grant; Italy grant; Italy grant; Italy grant; Italy grant; Japan UM; Japan app; Japan grant; Korea UM; Korea app; Korea grant; Latvia grant; Lithuania grant; Luxembourg app; Luxembourg grant; Malaysia app; Malaysia grant; Mexico app; Mexico grant; Moldova UM; Moldova grant; Monaco grant; Mongolia UM; Mongolia grant; Morocco app; Morocco grant; Netherlands app; Netherlands grant; New Zealand grant; Norway app; Norway grant; OAPI grant; PCT (WO) app; Philippines UM; Philippines app; Philippines grant; Poland UM; Poland grant; Portugal UM; Portugal app; Portugal grant; Romania UM; Romania app; Romania granted; Russia UM; Russia app; Russia grant; Serbia UM; Serbia app; Serbia grant; Singapore app; Singapore grant; Slovakia UM; Slovakia app; Slovakia grant; Slovenia UM; Slovenia grant; South Africa app; Spain UM; Spain app; Spain grant; Sweden app; Sweden grant; Switzerland app; Switzerland grant; Taiwan UM; Taiwan appl; Taiwan grant; Thailand UM; Thailand app; Thailand grant; Tunisia app; Turkey UM; Turkey app; Turkey grant; UK app; UK grant; USSR grant; Ukraine UM; Ukraine grant; United States app; United States grant; Uruguay app; Vietnam UM; Vietnam app; Vietnam grant; Other

Table of Contents

1. US6691139B2 Recreation of archives at a disaster recovery site

Family 1/1

1 record(s) per family

Record 1/1 US6691139B2 Recreation of archives at a disaster recovery site

Title: Recreation of archives at a disaster recovery site

Title - DWPI: Standby data processing system creation method for disaster recovery of computer system, involves transmitting active data file and delta image from primary site to standby site

Priority Number: US2001774839A

Priority Date: 2001-01-31

Application Number: US2001774839A

Application Date: 2001-01-31

Publication Date: 2004-02-10

IPC Class Table:

IPC	Section	Class	Subclass	Class Group	Subgroup
G06F001114	G	G06	G06F	G06F0011	G06F001114
G06F001120	G	G06	G06F	G06F0011	G06F001120

IPC Class Table - DWPI:

IPC - DWPI	Section - DWPI	Class - DWPI	Subclass - DWPI	Class Group - DWPI	Subgroup - DWPI
G06F001100 (IPC 1-7)	G	G06	G06F	G06F0011	G06F001100 (IPC 1-7)
G06F001200 (IPC 1-7)	G	G06	G06F	G06F0012	G06F001200 (IPC 1-7)

Assignee/Applicant: Hewlett Packard Development Co. | Ltd.,02

JP F Terms:

JP FI Codes:

Assignee - Original: Hewlett Packard Development Co. | Ltd.

Any CPC Table:

Type	CPC - Current - Invention	CPC - Current - Additional	Version	Office

Current	G06F 11/2071	G06F 11/1451	20130101	EP; US
Current	G06F 11/1464	G06F 11/2064	20130101	EP; US
Current	G06F 11/1471	G06F 11/2097	20130101	EP; US
Current	G06F 11/1474	G06F 2201/80	20130101	EP; US
Current	G06F 11/2066		20130101	EP; US
Current	G06F 11/2079		20130101	EP; US
Current	-	Y10S 707/99955	20130101	US
Original	G06F 11/2071	G06F 11/1451	20130101	US
Original	G06F 11/1464	G06F 11/2064	20130101	US
Original	G06F 11/1471	G06F 11/2097	20130101	US
Original	G06F 11/1474	G06F 2201/80	20130101	US
Original	G06F 11/2066	Y10S 707/99955	20130101	US
Original	G06F 11/2079		20130101	US

ECLA: G06F001120S2P | G06F001114A10H | G06F001120S2L | G06F001120S2P6 | S06F001114A10D2 | T05K099999 | S06F001120S2E | S06F001120U

Abstract:

A method for creating a standby data processing system located at a standby site, which is remotely located from a primary site. The method comprises a first step of computing a difference between an active data file and an archive data file to produce a delta image at the primary site. Another step is transmitting the active data file and the delta image, separately in time sequence, from the primary site to a standby site. A final step is combining the delta image with the active data file at the standby site, to produce an archive data file. An advantage of this method is the creation of a standby data processing system, while using a lower bandwidth to create this additional site.

Language of Publication: EN

INPADOC Legal Status Table:

Gazette Date	Code	INPADOC Legal Status Impact
2021-02-22	AS	-
Description: ASSIGNMENT VALTRUS INNOVATIONS LIMITED, IRELAND ASSIGNMENT OF ASSIGNORS INTEREST; ASSIGNORS:HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP; HEWLETT PACKARD ENTERPRISE COMPANY; REEL/FRA ME:055360/0424 2021-01-21		
2016-02-10	SULP	+
Description: SURCHARGE FOR LATE PAYMENT FEE PAYMENT YEAR 11		
2016-02-10	FPAY	+

Description: FEE PAYMENT FEE PAYMENT YEAR 12		
2015-11-09	AS	-
Description: ASSIGNMENT HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP, TEXAS ASSIGNMENT OF ASSIGNORS INTEREST; ASSIGNOR:HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P.; REEL/FRAME:037079/0001 2015-10-27		
2015-09-18	REMI	-
Description: MAINTENANCE FEE REMINDER MAILED		
2011-08-10	FPAY	+
Description: FEE PAYMENT FEE PAYMENT YEAR 8		
2007-08-10	FPAY	+
Description: FEE PAYMENT FEE PAYMENT YEAR 4		
2004-01-22	STCF	-
Description: INFORMATION ON STATUS: PATENT GRANT PATENTED CASE		
2003-09-30	AS	-
Description: ASSIGNMENT HEWLETT-PACKARD DEVELOPMENT COMPANY L.P., TEXAS ASSIGNMENT OF ASSIGNORS INTEREST; ASSIGNOR:HEWLETT-PACKARD COMPANY; REEL/FRAME:014061/0492 2003-09-26		
2003-09-30	AS	-
Description: ASSIGNMENT HEWLETT-PACKARD DEVELOPMENT COMPANY L.P.,TEXAS ASSIGNMENT OF ASSIGNORS INTEREST; ASSIGNOR:HEWLETT-PACKARD COMPANY; REEL/FRAME:014061/0492 2003-09-26		
2001-05-21	AS	-
Description: ASSIGNMENT HEWLETT-PACKARD COMPANY, COLORADO ASSIGNMENT OF ASSIGNORS INTEREST; ASSIGNORS:GANESH, SHIVAJI; KAVANAPPILLIL, RAMESH; REEL/FRAME:011830/0946; SIGNING DATES FROM 20010418 TO 20010426		

Post-Issuance (US):

Reassignment (US) Table:

Assignee	Assignor	Date Signed	Reel/Frame	Date
VALTRUS INNOVATIONS LIMITED,DUBLIN,IE	HEWLETT PACKARD ENTERPRISE	2021-01-21	055360/0424	2021-02-22

	DEVELOPMENT LP			
	HEWLETT PACKARD ENTERPRISE COMPANY	2021-01-21	-	-
Conveyance: ASSIGNMENT OF ASSIGNORS INTEREST (SEE DOCUMENT FOR DETAILS).				
Corresponent: BOTOS CHURCHILL IP LAW LLP 430 MOUNTAIN AVENUE, SUITE 401 NEW PROVIDENCE, NJ 07974				
HEWLETT PACKARD ENTERPRISE DEVELOPMENT LP,HOUSTON,TX,US	HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P.	2015-10-27	037079/0001	2015-11-09
Conveyance: ASSIGNMENT OF ASSIGNORS INTEREST (SEE DOCUMENT FOR DETAILS).				
Corresponent: HEWLETT PACKARD ENTERPRISE 3404 E. HARMONY ROAD MS 79 FORT COLLINS, CO 80528				
HEWLETT-PACKARD DEVELOPMENT COMPANY L.P.,HOUSTON,TX,US	HEWLETT-PACKARD COMPANY	2003-09-26	014061/0492	2003-09-30
Conveyance: ASSIGNMENT OF ASSIGNORS INTEREST (SEE DOCUMENT FOR DETAILS).				
Corresponent: HEWLETT-PACKARD COMPANY GUY J. KELLEY P.O. BOX 272400 INTELLECTUAL PROPERTY ADMINISTRATION FORT COLLINS, CO 80527-2400				
HEWLETT-PACKARD COMPANY,FORT COLLINS,CO,US	GANESH, SHIVAJI	2001-04-18	011830/0946	2001-05-21
	KAVANAPPILLIL, RAMESH	2001-04-26	-	-
Conveyance: ASSIGNMENT OF ASSIGNORS INTEREST (SEE DOCUMENT FOR DETAILS).				
Corresponent: RECORDS MANAGER HEWLETT-PACKARD COMPANY MARCI SANTILLANES LEGAL DEPT MS 79, 3404 E. HARMONY ROAD FT. COLLINS, CO 80528				

Maintenance Status (US):

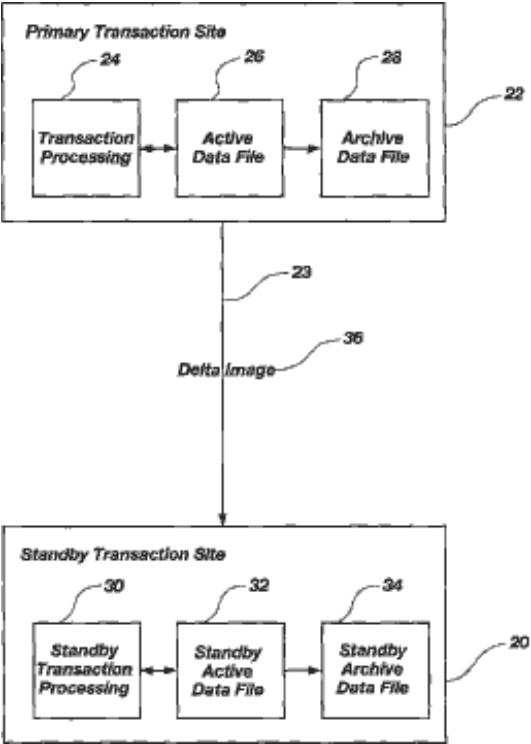
Litigation (US):

Opposition (EP):

License (EP):

EPO Procedural Status:

Front Page Drawing:



Priority Number - DWPI: US2001774839A
Priority Date - DWPI: 2001-01-31
Assignee - Current US: VALTRUS INNOVATIONS LIMITED



United States Patent and Trademark Office

Office of the Commissioner for Patents

RECREATION OF ARCHIVES AT A DISASTER RECOVERY SITE

PATENT #

6691139

APPLICATION #

09774839

FILING DATE

01/31/2001

ISSUE DATE

02/10/2004

Payment Window Status

WINDOW	STATUS		FEES		
11.5 Year	Closed		Paid		
Window	First Day to Pay	Surcharge Starts	Last Day to Pay	Status	Fees
3.5 Year	02/10/2007	08/11/2007	02/11/2008	Closed	Paid
7.5 Year	02/10/2011	08/11/2011	02/10/2012	Closed	Paid
11.5 Year	02/10/2015	08/11/2015	02/10/2016	Closed	Paid

No maintenance fees are due.

Patent Holder Information

Customer #

Entity Status UNDISCOUNTED

Phone Number

Address HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, CO 80527-2400
UNITED STATES