UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

AMERICAN AIRLINES, INC. AND SOUTHWEST AIRLINES CO.,

Petitioners

v.

INTELLECTUAL VENTURES II LLC

Patent Owner.

IPR2025-00931

U.S. Patent No. 8,332,844

DECLARATION OF EREZ ZADOK IN SUPPORT OF PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 8,332,844

Table of Contents

I. ASSIGNMENT 1

II. BACKGROUND AND QUALIFICATIONS 1

III. MATERIALS CONSIDERED 12

IV. LEVEL OF ORDINARY SKILL IN THE ART 15

V. RELEVANT LEGAL STANDARDS 16

VI. SUMMARY OF OPINIONS 20

VII. TECHNOLOGY OVERVIEW 21

- **A.** General Computer Operations 21
  - 1. Networking Overview 23
- **B.** Data Storage Devices and File Systems 27
  - 1. Files and File Systems 28
- **C.** Networked and Distributed Storage Systems 32
- **D.** Caching Technologies 36
  - 1. Least-Recently Used (LRU) 44
  - 2. Least-Frequently Used (LFU) 44
- **E.** Copy-on-Write and Snapshots 46
- **F.** Indexing 58
- **G.** Virtualization 59

VIII. THE ’844 PATENT 63

- **A.** Overview 63
- **B.** The ’844 Patent Discloses Indexing at the Block Layer 73

C. The ’844 Patent’s Union Block Device (UBD) 76

D. Prosecution History 78

1. The ’477 Application 78

2. The ’622 Application 84

3. Priority Date 86

IX. THE CLAIMS 87

X. OVERVIEW OF THE PRIMARY PRIOR ART REFERENCES.....94

A. Sapuntzakis.....95

B. Holzmann 102

C. Federwisch.....107

D. Birse.....112

XI. CLAIM CONSTRUCTION 122

XII. DETAILED EXPLANATION OF GROUNDS.....123

A. Ground 1: Sapuntzakis In View Of Holtzman Renders Claim 1-13 Obvious. 123

1. Claim 7 123

a. [7.0] A method for providing data to a plurality of compute nodes, comprising: 123

b. [7.1] storing blocks of a root image of said compute nodes on a first storage unit; 124

c. [7.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do

not include blocks of said root image that are unchanged by respective compute nodes; and 126

d. [7.3] caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory. 129

2. Motivation to Combine Sapuntzakis and Holzmann 133

3. Claim 8 139

a. [8.1] The method as recited in claim 7 further comprising: receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and 139

b. [8.2] providing said first portion of said data to said at least one of said compute nodes from said cache memory. 140

4. Claim 9: The method as recited in claim 8 further comprising: updating said cache memory based on said read request. 142

5. Claim 10 143

a. [10.0] The method as recited in claim 9 wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises:.....143

b. [10.1] caching said second portion in said cache memory; and.....144

c. [10.2] removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value. 145

6. Claim 11: The method as recited in claim 7 further comprising: merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments. 147

7. Claim 12: The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory. 149

8. Claim 13: The method as recited in claim 7 wherein said first storage unit is read-only. 150

9. Claim 1 151

   a. [1.0] A system for providing data to a plurality of compute nodes, the system comprising: 151

   b. [1.1] a first storage unit configured to store blocks of a root image of said compute nodes; 151

   c. [1.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and.152

   d. [1.3] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes. 152

10. Claim 2: The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value. 152

11. Claim 3: The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance. 154

12. Claim 4: The system as recited in claim 1 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block

devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images. 156

13. Claim 5: The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.....157

14. Claim 6: The system as recited in claim 1 wherein said first storage unit is read-only. 157

B. Ground 2: Sapuntzakis In View Of Federwisch Renders Claim 14-27 Obvious. 157

1. Claim 14 157

a. [14.0] A system for indexing file systems for a plurality of compute nodes, the system comprising:.....157

b. [14.1] a first storage unit configured to store blocks of a root image of said compute nodes; 157

c. [14.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and.158

d. [14.3]-[14.4] a plurality of union block devices corresponding to said compute nodes, said union block devices configured to interface between said compute nodes and said first and second storage units to distribute said file systems to said compute nodes, wherein said union block devices are configured to create said file systems by merging the blocks of said root image stored on the first storage unit with the blocks of respective leaf images stored on respective second storage units, and

wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.....158

2. Motivation to Combine Sapuntzakis with Federwisch.....162

3. Claim 15: The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance. 168

4. Claim 16: The system as recited in claim 14 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit and respective second storage units, said union block devices configured to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images. 168

5. Claim 17: The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.....169

6. Claim 18: The system as recited in claim 14 wherein said first storage unit is read-only.....169

7. Claim 19.....169

   a. [19.0] A method for indexing file systems for a plurality of compute nodes, comprising:.....169

   b. [19.1] storing blocks of a root image of said compute nodes on a first storage unit;.....169

   c. [19.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image,

wherein said leaf images for respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; 169

d. [19.3] merging the blocks of said root image with the blocks of respective leaf images stored on respective second storage units to create respective file systems for respective compute nodes;.....170

e. [19.4] receiving indexing results pertaining to said root image from one of said compute nodes; and.....170

f. [19.4] providing said indexing results to the others of said compute nodes.....170

8. Claim 20: The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit.....170

9. Claim 21: The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the compute nodes and said first storage unit and respective second storage units.....171

10. Claim 22: The method as recited in claim 19 wherein said first storage unit is read-only.....172

11. Claim 23 172

a. [23.0] A computer-readable storage medium having instructions stored thereon that, in response to execution by at least one computing device, cause the at least one computing device to: 172

b. [23.1]-[23.3] receive data blocks of a file system, said data blocks comprising a root image portion and leaf image portion, said leaf image portion comprising only additional data blocks not previously contained in said root image portion and changes made by said first compute node to the blocks of said root image, wherein said leaf image portion does not include blocks of said root image that are unchanged by said first compute

node, wherein said file system is the result of merging said root image portion and said leaf image portion together at the block-level; 172

c. [23.4] index said root image portion; and 172

d. [23.4] provide the results of said indexing to a second compute node, wherein said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node. 173

12. Claim 24: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to store said results of said indexing on a shared storage unit accessible by said second compute node.173

13. Claim 25: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion. 173

14. Claim 26: The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion. 175

15. Claim 27: The computer-readable storage medium of claim 23 wherein said root image portion is read-only. 177

C. Ground 3: Birse In View Of Holtzman Renders Claim 1-13 Obvious. 177

1. Claim 7 178

a. [7.0] A method for providing data to a plurality of compute nodes, comprising: 178

b. [7.1] storing blocks of a root image of said compute nodes on a first storage unit; 180

c. [7.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images

including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and 183

d. [7.3] caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory. 187

2. Motivation to Combine Birse and Holzmann 188

3. Claim 8 193

a. [8.1] The method as recited in claim 7 further comprising: receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and 193

b. [8.2] providing said first portion of said data to said at least one of said compute nodes from said cache memory. 193

4. Claim 9: The method as recited in claim 8 further comprising: updating said cache memory based on said read request. 194

5. Claim 10 194

a. [10.0] The method as recited in claim 9 194

b. [10.1] wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises: caching said second portion in said cache memory; and 194

c. [10.2] removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value. 194

6. Claim 11: The method as recited in claim 7 further comprising: merging the blocks of said root image with the blocks of

respective leaf images to create cohesive respective application environments. 195

7. Claim 12: The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory. 197

8. Claim 13: The method as recited in claim 7 wherein said first storage unit is read-only. 198

9. Claim 1 199

   a. [1.0] A system for providing data to a plurality of compute nodes, the system comprising: 199

   b. [1.1] a first storage unit configured to store blocks of a root image of said compute nodes; 199

   c. [1.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and.199

   d. [1.3] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes. 199

10. Claim 2: The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value. 200

11. Claim 3: The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance. 201

12. Claim 4: The system as recited in claim 1 further comprising: a plurality of union block devices configured to interface between

respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images. 203

13. Claim 5: The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.....204

14. Claim 6: The system as recited in claim 1 wherein said first storage unit is read-only. 204

D. Ground 4: Birse In View Of Federwisch Renders Claim 14-27 Obvious. 205

1. Claim 14 205

a. [14.0] A system for indexing file systems for a plurality of compute nodes, the system comprising:.....205

b. [14.1] a first storage unit configured to store blocks of a root image of said compute nodes; 205

c. [14.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and.205

d. [14.3]-[14.4] a plurality of union block devices corresponding to said compute nodes, said union block devices configured to interface between said compute nodes and said first and second storage units to distribute said file systems to said compute nodes, wherein said union block devices are configured to create said file

systems by merging the blocks of said root image stored on the first storage unit with the blocks of respective leaf images stored on respective second storage units, and wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.....206

2. Motivation to Combine Birse with Federwisch.....209

3. Claim 15: The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance. 217

4. Claim 16: The system as recited in claim 14 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit and respective second storage units, said union block devices configured to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images. 217

5. Claim 17: The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.....218

6. Claim 18: The system as recited in claim 14 wherein said first storage unit is read-only.....218

7. Claim 19.....218

   a. [19.0] A method for indexing file systems for a plurality of compute nodes, comprising:.....218

   b. [19.1] storing blocks of a root image of said compute nodes on a first storage unit;.....218

   c. [19.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images

comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images for respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; 218

d. [19.3] merging the blocks of said root image with the blocks of respective leaf images stored on respective second storage units to create respective file systems for respective compute nodes;.....219

e. [19.4] receiving indexing results pertaining to said root image from one of said compute nodes; and.....219

f. [19.4] providing said indexing results to the others of said compute nodes.....219

8. Claim 20: The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit.....219

9. Claim 21: The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the compute nodes and said first storage unit and respective second storage units.....220

10. Claim 22: The method as recited in claim 19 wherein said first storage unit is read-only.....221

11. Claim 23 221

a. [23.0] A computer-readable storage medium having instructions stored thereon that, in response to execution by at least one computing device, cause the at least one computing device to: 221

b. [23.1]-[23.3] receive data blocks of a file system, said data blocks comprising a root image portion and leaf image portion, said leaf image portion comprising only additional data blocks not previously contained in said root image portion and changes made by said first

compute node to the blocks of said root image, wherein said leaf image portion does not include blocks of said root image that are unchanged by said first compute node, wherein said file system is the result of merging said root image portion and said leaf image portion together at the block-level; 221

c. [23.4] index said root image portion; and 221

d. [23.4] provide the results of said indexing to a second compute node, wherein said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node. 222

12. Claim 24: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to store said results of said indexing on a shared storage unit accessible by said second compute node.222

13. Claim 25: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion. 222

14. Claim 26: The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion. 224

15. Claim 27: The computer-readable storage medium of claim 23 wherein said root image portion is read-only. 226

XIII. SECONDARY CONSIDERATIONS 226

XIV. CONCLUSION.....226

I, Erez Zadok, Ph.D., declare that:

I. ASSIGNMENT

1. I have been retained by American Airlines, Inc. and Southwest Airlines Co. (collectively "Petitioners") as an independent expert consultant in this proceeding before the United States Patent and Trademark Office ("PTO").

2. My consulting company, Zadoks Consulting Services, is being compensated for my time at my standard consulting rate. I am also being reimbursed for expenses that I may incur during the course of this work.

3. My compensation is in no way contingent on the nature of my findings, the presentation of my findings in testimony, or the outcome of this or any other proceeding. I have no other interest in this proceeding.

4. I have been asked to consider whether certain references disclose or suggest the features recited in the claims of U.S. Patent No. 8,332,844 (the "'844 patent") (EX1001).[1] My opinions are set forth below.

II. BACKGROUND AND QUALIFICATIONS

5. I am a Professor in the Computer Science Department at Stony Brook University (part of the State University of New York ("SUNY") system). I direct

[1] Where appropriate, I refer to exhibits I understand will be attached to the petition for inter partes review of the '844 Patent (the "Petition").

the File-systems and Storage Lab (FSL) at Stony Brook's Computer Science Department. My research interests include file systems and storage systems, operating systems, transactional systems including database technologies, information technology and system administration, security/privacy and information assurance, networking, energy efficiency, performance and benchmarking, virtualization, cloud systems, compilers, applied machine learning, and software engineering.

6. I studied at a professional high school in Israel, focusing on electrical engineering ("EE"), and graduated in 1982. I spent one more year at the high school's college division, receiving a special Certified Technician's degree in EE. I then went on to serve in the Israeli Defense Forces for three years (1983–1986). I received my Bachelor of Science degree in computer science ("CS") in 1991, my Master's degree in CS in 1994, and my PhD in CS in 2001—all from Columbia University in New York.

7. When I began my undergraduate studies at Columbia University, I also started working as a student assistant in the various campus-wide computer labs, eventually becoming an assistant to the head labs manager, who was managing all public computer labs on campus. During that time, I also became more involved with research within the CS Department at Columbia University, conducting research on operating systems, file and storage systems, distributed and networked

systems, security, and other topics. I also assisted the CS department's computer administrators in managing the department's computers, which included storage, IT, networking, and cyber-security related duties.

8. In 1991, I joined Columbia University's CS department as a full-time systems administrator, studying towards my MS degree part-time. My MS thesis topic is related to file system reliability, fault tolerance, replication, and failover in mobile networked storage systems using file virtualization. My main duties as a systems administrator involved installing, configuring, and managing many networked servers, proxies, and desktops running several operating systems, as well as network devices setup; this included many software and hardware upgrades, device upgrades, and BIOS firmware/chipset updates/upgrades. My duties also included ensuring reliable, secure, authenticated access to networked systems/storage and licensed software, as well as software updates, security and bug fixes. Examples of servers and their protocols included email (SMTP), file transfer (FTP), domain names (DNS), network file systems (NFS), network news systems (NNTP), and Web (HTTP).

9. In 1994, I left my systems administrator position to pursue my doctoral studies at Columbia University. My PhD thesis topic was on versatile file system development using stackable (virtualized) file systems, with examples in the fields of security and encryption, efficiency, reliability, and failover. I continued to work

part-time as a systems administrator at the CS department, and eventually I was asked to serve as manager to the entire information technology ("IT") staff. From 1991 to 2001, I was also a member of the faculty-level Facilities Committee that oversaw all IT operations at the CS department.

10. As part of my PhD studies at Columbia, I collaborated on projects to develop advanced AI-like techniques to detect previously unknown viruses (a.k.a. "zero-day malware"), using data mining and rule-based detection. This work led to several highly cited papers (over 1,600 citations for one of the papers alone) and two patents. I also became a Teaching Assistant ("TA") for a first-ever Computer Security course given at Columbia University's CS department with Dr. Matt Blaze as instructor.

11. From 1990 to 1998, I consulted for SOS Corporation and HydraWEB Technologies as a systems administrator and programmer, managing data storage use and backup/restore duties, databases, web servers, as well as information assurance and cyber-security (*e.g.*, malware protection, software licensing). From 1994 to 2000, I led projects at HydraWEB Technologies, and then became the Director of Software Development—overseeing the development of several products and appliances such as stateful firewalls and HTTP load-balancers, utilizing network-virtualization and high-availability techniques. From 2009 to

2019, I have consulted for Packet General Networks, a startup specializing in secure, virtualized, network storage and applications' data security in the cloud.

12. In 2001, I joined the faculty of Stony Brook University, a position I have held since that time. In 2002, I joined the Operations Committee, which oversees the IT operations of the CS department at Stony Brook University. From 2006 to 2010, I was the Director of IT Operations of the CS department. My day-to-day duties included setting policies regarding computing, hiring and training new staff, assisting any staff with topics of my specialty, defining requirements for new software/hardware, and purchasing. From 2010 to 2015, I had served as the Co-Chair to the Operations Committee. From 2016 to 2019, I oversaw the IT Operations as the Chair of the Operations Committee. A significant component of these duties included defining and helping implement policies for data management, to ensure the security of users and their data, and data reliability and availability, while minimizing the inconvenience and performance impact to users. I personally helped setup and maintain an initial virtual-host infrastructure in the department. Since late 2019, I've been a member of the department's Executive Committee that also oversees all IT operations.

13. In 2017, I became the department's Graduate Academic Adviser, advising all Master students (over 400 annually on average) and many other graduate students on an assortment of academic matters. In August 2024, I took over as the

department's Graduate Program Director, overseeing the entire graduate CS program (700-800 students annually on average).

14. Since 2001, I have personally configured and managed my own research lab's network. This includes setting up and configuring multiple storage systems (*e.g.*, NFS, CIFS/SMB, NAS), virtual and physical environments, applications such as database (*e.g.*, MySQL, PostgreSQL), Web servers (*e.g.*, Apache), and mail servers; user access control (*e.g.*, NIS, LDAP), backups and restores, snapshot policies, and more. I've personally installed, configured, changed, replaced parts, and upgraded components in numerous devices including mobile devices, laptops, desktops, and servers, both physical and virtual.

15. Since 1995, I have taught courses on operating systems, storage and file systems, advanced systems programming in Unix/C, systems administration, data structures, data/software security, and more. My courses often use storage, file systems, distributed systems, and system/network security as key teaching principles and practical examples for assignments and projects. I have taught these concepts and techniques to my students, both to my direct advisees as well as in my courses. For example, in my graduate Operating Systems course, I often cover Linux's kernel mechanisms to protect users, applications, and data files, virtual file systems, as well as distributed storage systems (*e.g.*, NFS). And in the System Administration

undergraduate course, I covered many topics such as networking, storage, backups, and configuring complex applications such as mail, web, and database servers.

16. My research often investigates computer systems from many angles: security, efficiency, energy use, scalability, reliability, portability, survivability, usability, ease-of-use, versatility, flexibility, and more. My research gives special attention to balancing five often-conflicting aspects of computer systems: performance, reliability, energy use, security, and ease-of-use.

17. Since joining Stony Brook University in 2001, my group in the File-systems and Storage Lab ("FSL") has developed many file systems and operating system extensions; examples include a highly-secure cryptographic file system, a portable copy-on-write ("COW") versioning file system, a tracing file system useful to detect intrusions, a replaying file system useful for forensics, a snapshotting and sandboxing file system, a namespace unification file system (that uses stackable, virtualized, file-based COW), an anti-virus file system, an integrity-checking file system, a load balancing and replication/mirroring file system, network file system extensions for security and performance, distributed secure cloud-based storage systems, transactional key-value stores and file systems, OS-level embedded databases, a compiler to convert user-level C code to in-kernel efficient yet safe code, GCC plugins, stackable file system templates, and a Web-based backup system. Many of these projects used one form of virtualization or another (storage,

network, host, etc.). I continue to maintain and release newer versions of some of these file systems and software.

18. I have published over 120 refereed publications (in ACM, IEEE, USENIX, and more). To date, my publications have been cited more than 10,000 times (as per Google Scholar as of March 27. 2025). My papers cover a wide range of related technologies such as file systems, storage systems, transactional systems, security, clouds and virtualization, performance benchmarking and optimization, energy efficiency, system administration, web systems, and more. I also published a book titled "Linux NFS and Automounter Administration" (Sybex, 2001), covering systems administration topics related to network storage and data security.

19. Some of my research has led to public software releases that have been used worldwide. I have publicly maintained the Amd Berkeley Automounter in a package called "am-utils" since 1992; this software helps administrators manage the multitude of file system mounts on dozens of different Unix systems, especially helping to automate access to multiple NFS/NAS storage volumes. Since 1997, I have maintained and released several stackable (virtualized) file system software projects for Linux, FreeBSD, and/or Sun Solaris, in a package called FiST. One of my stackable file system encryption projects, called Cryptfs, became the basis for IBM's public release of eCryptfs, now part of Linux. Packet General Networks, for whom I have provided consulting services between 2009 and 2019, licensed another

encryption file system called Ncryptfs. Another popular file system released in 2003, called Unionfs, offers virtual namespace unification, transparent shadow copying (a.k.a. copy-on-write or COW), file system snapshotting (*e.g.*, useful for forensics and disaster recovery), and the ability to save disk space by sharing a read-only copy of data among several computers, among other features.

20. My research and teaching make extensive use of data security features. For example, each time I taught the graduate operating system course, the first homework assignment includes the creation of a new system call that performs new or added functionality, often for encrypting a file or verifying its integrity; many of my other assignments cover topics of user/process access control, anti-virus filtering, and more. Since 2001, over 1,000 graduate students were exposed to these principles directly through my teaching and research at Stony Brook University.

21. Moreover, in an undergraduate course titled "Advanced Systems Programming in Unix/C," I cover many topics of system security and vulnerabilities, such as the structure of UNIX processes, and memory segments such as the heap and stack. This course covers details of several hundred Linux system calls. Often, the first assignment for this course is to develop a tool to encrypt/decrypt files using advanced ciphers, use digital signatures to certify the cipher keys used, and reliably recover files in case of failures. Since 2001, several hundred undergraduate students

were exposed to these principles directly through my teaching and research at Stony Brook University.

22. In another undergraduate course, System Administration, I taught network configuration, security, and storage configuration and reliability. In a special topics course on Storage Systems, I covered many topics such as data deduplication, RAID, transactional storage, storage hardware including modern Flash based ones, virtual storage, backup/restore, snapshots and continuous data protection ("CDP"), NAS and SAN, and NFS.

23. Overall, in addition to the aforementioned experience, my technical experience relevant to the '844 Patent at the time of the alleged invention included the configuration, use, research, development, and/or programming of the following technologies and systems: data storage devices, drivers, and services; operating systems layers including (virtual) block layer, file systems, and virtual file systems; copy-on-write storage system and file systems (e.g., Unionfs), including snapshots; commercial and open-source storage appliances (including from Netapp); and caching techniques in one or multi-tier systems.

24. My research has been supported by many federal and state grants as well as industry awards, including an NSF CAREER award, two IBM Faculty awards, two NetApp Faculty awards, a Western Digital award, a Facebook award, several Dell-EMC awards, and several equipment gifts. I received the 2008 SUNY

Chancellor's Excellence in Teaching award, and the 2022 SUNY Chancellor's Award for Excellence in Scholarship and Creative Activities (both awards can be given only once in a lifetime). In 2021, I was named an ACM Distinguished Member for "Outstanding Scientific Contributions to Computing."

25. My service record to the community includes serving as the co-chair for the USENIX Annual Technical Conference in 2020 (ATC'20); serving as the co-chair for USENIX File and Storage Technologies (FAST'15) in 2015 and on the FAST Conference Steering Committee from 2015 to 2023; serving on the ACM HotStorage Steering Committee since 2021; and serving as the co-chair in 2012 and on the Steering Committee of the ACM SYSTOR conference since 2012. I have served as an Associate Editor to the ACM Transactions on Storage ("TOS") journal from 2009 to 2022; in 2022, I was named the Editor-in-Chief for ACM's TOS journal.

26. I am a named inventor on four patents, two titled "Systems and Methods for Detection of New Malicious Executables" (U.S. Patent No. 7,487,544, issued February 3, 2009; and U.S. Patent No. 7,979,907, issued July 12, 2011); and two more titled "Multi-Tier Caching," (U.S. Patent No. 9,355,109, issued May 31, 2016; and U.S. Patent 9,959,279, issued May 1, 2018).

27. I have been disclosed as a testifying expert in 19 cases (including *inter partes* review ("IPR") proceedings) in the past four years. I have been deposed 13 times and testified in trial twice.

28. A copy of my *curriculum vitae* is attached as EX1004.

III. MATERIALS CONSIDERED

29. The opinions contained in this declaration are based on the documents I reviewed, my professional judgment, as well as my education, experience, and knowledge regarding electrical engineering, computer science, video and audio and communications sciences and technologies.

30. In forming my opinions expressed in this declaration, I reviewed the following materials:

Exhibit	Description
1001	U.S. Patent No. 8,332,844 B1, entitled "Root Image Caching and Indexing for Block-Level Distributed Application Management," issued December 11, 2012 to Kulkarni, et al. ("the '844 patent")
1002	File History for U.S. Patent No. 8,332,844
1004	Dr. Erez Zadok Curriculum Vitae
1005	Sapuntzakis, et al. "Optimizing the Migration of Virtual Computers," Association for Computing Machinery, Special Interest Group on Operating Systems, Operating Systems Review, Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI), Volume 36, Issue S1, published December 31, 2002 ("Sapuntzakis")
1006	U.S. Patent No. 7,089,300B1, entitled "Method and Apparatus for Administering the Operating System of a Net-Booted

Exhibit	Description
	Environment," filed on October 18, 1999, and issued August 8, 2006 to Birse, et al. ("Birse")
1007	U.S. Patent Application Publication No. 2005/0283597 A1, entitled "System and Method for Booting Multiple Servers from a Single Operating System Image," filed on June 22, 2004, and published on December 22, 2005 (matured into U.S. Patent No. 8,601,100, issued December 3, 2013 to Holzmann) ("Holzmann")
1008	U.S. Patent Application Publication No. 2003/0182313 A1, entitled "System and Method for Determining Changes in Two Snapshots and for Transmitting Changes to Destination Snapshot," filed on March 19, 2002, and published on September 25, 2003 (matured into U.S. Patent No. 6,993,539, issued January 31, 2006 to Federwisch, et al.) ("Federwisch")
1009	File History for U.S. Patent Application No. 11/026,622 ("the 622 Application", now abandoned)
1010	File History for U.S. Patent Application No. 11/395,816 ("the 816 Application", now U.S. Patent No. 7,721,282)
1011	Microsoft Computer Dictionary, 3rd ed., 1997, excerpts ("Microsoft Computer Dictionary")
1012	Abraham Silberschatz and Peter B. Galvin, Operating Systems Concepts, 4th Edition, 1994, Addison-Wesley Publishing, excerpts ("Silberschatz")
1013	File System Design for an NFS File Server Appliance, Dave Hitz et al., Network Appliance, USENIX 1994 ("Hitz")
1014	SnapMirror: File System Based Asynchronous Mirroring for Disaster Recovery, Hugo Patterson et al., Network Appliance, USENIX 2002 ("Patterson I")
1015	U.S. Patent 6,668,264B1 to Patterson et al., Filed Apr. 3, 2001, Issued Dec. 23, 2003, titled "Resynchronization of a target volume with a source volume" ("Patterson II")
1016	U.S. Patent 7,334,095B1 to Fair et al. Filed Apr. 30, 2004 ("Fair")

Exhibit	Description
1017	Internet Small Computer Systems Interface (iSCSI), RFC 3720, IETF, April 2004, https://www.rfc-editor.org/rfc/pdfrfc/rfc3720.txt.pdf ("RFC-3720")

31. Exhibits 1011-1014 and 1017 are true and correct copies of the identified documents that I personally retrieved, copied, own, and/or scanned for purposes of submission with this IPR proceeding. I have also reviewed any other materials I refer to in this declaration in support of my opinions.

32. All of the opinions contained in this declaration are based on the documents I reviewed and my knowledge, experience, and professional judgment. My opinions have also been guided by my appreciation of how a person of ordinary skill in the art would have understood the claims and the specification of the '844 Patent at the time of the alleged invention, which I have been asked to initially consider as no earlier than December 30, 2004 (the filing date of U.S. Patent Application 11/026,622, EX1009). My opinions reflect how one of ordinary skill in the art would have understood the '844 Patent, the prior art to the patent, and the state of the art at the time of the alleged invention.

33. Based on my experience and expertise, it is my opinion that certain references disclose and/or suggest all the features recited in claims 1-27 ("challenged claims") of the '844 Patent, as I discuss in detail below.

IV. LEVEL OF ORDINARY SKILL IN THE ART

34. I have been informed and understand that, in the context of an invalidity analysis, a person having ordinary skill in the art ("POSITA") is not a specific real individual, but a hypothetical person who looks to prior art at the time of the invention. I further understand that the factors that may be considered in determining the level of ordinary skill include: (1) the level of education and experience of persons working in the field; (2) the types of problems encountered in the field; (3) the teachings of the prior art regarding solutions to such problems; (4) the sophistication of the technology at the time of the alleged invention; and (5) the rapidity of innovation. I understand that these factors need not all be considered for the analysis, and that one or more of these factors may control.

35. I was asked to provide my opinion on the level of one of ordinary skill in the art with respect to the alleged invention of the '844 patent no earlier than December 30, 2004. In my opinion, a POSITA would have had a bachelor's degree in computer science, computer engineering, or an equivalent degree. A POSITA would also have had approximately two years of experience working in the fields of networked data storage, file systems, and operating systems. Additional experience might substitute for less education and *vice versa*. It is also my opinion that these qualifications would have been the same on March 30, 2006 (the filing date of U.S.

Application No. 11/395,816) and February 21, 2007 (the filing date of the '844 Patent).

36. As of December 30, 2004, I met, and in fact exceeded, the qualifications of a person of ordinary skill in the art. To be clear, all of my opinions in this declaration are from the perspective of one of ordinary skill in the art as I have defined it here during the relevant timeframe.

V. RELEVANT LEGAL STANDARDS

37. I am not an attorney and offer no legal opinions, but in the course of my work I have had experience studying and analyzing patents and patent claims from the perspective of a person skilled in the art.

38. For the purposes of my analysis and conclusions expressed in this declaration, I have applied the legal principles described in the following paragraphs.

39. Petitioners' counsel has informed me, and I understand, that for the prior art to inherently disclose a claimed limitation, the prior art need not expressly disclose the limitation, so long as the claimed limitation necessarily flows from a disclosure in the prior art. I also understand that it is acceptable to examine evidence outside the prior art reference (extrinsic evidence) in determining whether a feature, while not expressly discussed in the reference, is necessarily present in that reference.

40. Petitioners' counsel has informed me, and I understand, that a patent claim can be considered to have been obvious to a POSITA at the time the application was filed. I am informed that this means that, even if all of the requirements of a claim are not found in a single prior art reference, the claim is not patentable if the differences between the subject matter in the prior art and the subject matter in the claim would have been obvious to a POSITA at the time of the invention.

41. I have been informed by Petitioners' counsel and understand that a determination of whether a claim would have been obvious should be based upon several factors, including, among others:

- The level of ordinary skill in the art at the time the application was filed;
- The scope and content of the prior art; and
- What differences, if any, existed between the claimed invention and the prior art.

42. I have been informed by Petitioners' counsel and understand that a single reference can render a patent claim obvious if any differences between that reference and the claims would have been obvious to a POSITA. Alternatively, I understand that the teachings of two or more references may be combined in the same way as disclosed in the claims, if such a combination would have been obvious

to one having ordinary skill in the art. In determining whether a combination based on either a single reference or multiple references would have been obvious, it is appropriate to consider, among other factors:

- Whether the teachings of the prior art references disclose known concepts combined in familiar ways, and when combined, would yield predictable results;
- Whether a person having ordinary skill in the art could implement a predictable variation, and would see the benefit of doing so;
- Whether the claimed elements represent one of a limited number of known design choices, and would have a reasonable expectation of success by those skilled in the art;
- Whether a person having ordinary skill would have recognized a reason to combine known elements in the manner described in the claim;
- Whether there is some teaching or suggestion in the prior art to make the modification or combination of elements claimed in the patent; and
- Whether the innovation applies a known technique that had been used to improve a similar device or method in a similar way.

43. I have been informed by Petitioners' counsel and understand that one of ordinary skill in the art has ordinary creativity and is not an automaton. Petitioners' counsel has also informed me that in considering obviousness, obviousness may not be determined using the benefit of hindsight, including hindsight derived from the patent being considered.

44. I further understand that a hypothetical POSITA is assumed to know and to have all relevant prior art in the field of endeavor covered by the patent at issue. I have been informed that there are at least two criteria for determining whether prior art is analogous and thus can be considered prior art: (1) whether the art is from the same field of endeavor, regardless of the problem addressed; and (2) if the reference is not within the field of the patentee's endeavor, whether the reference still is reasonably pertinent to the particular problem with which the patentee is involved. I have also been informed that the field of endeavor of a patent is not limited to the specific point of novelty, the narrowest possible conception of the field, or the particular focus within a given field. I have also been informed that a reference is reasonably pertinent if, even though it may be in a different field from that of the patentee's endeavor, it is one which, because of the matter with which it deals, logically would have commended itself to a patentee's attention in considering their problem.

45. I further understand that, in making a determination as to whether the claimed invention would have been obvious to a POSITA, the U.S. Patent Trial and Appeal Board may consider certain objective factors if they are present, such as: commercial success of products practicing the claimed invention; long-felt but unsolved need; teaching away; unexpected results; copying; and praise by others in the field. These factors are generally referred to as "secondary considerations" or "objective indicia" of nonobviousness. I understand, however, that for such objective evidence to be relevant to the obviousness of a claim, there must be a causal relationship (called a "nexus") between the claim and the evidence, and that this nexus must be based on a novel element of the claim rather than something in the prior art. I also understand that even when they are present, secondary considerations may be unable to overcome primary evidence of obviousness (such as motivation to combine with predictable results) that is sufficiently strong.

VI. SUMMARY OF OPINIONS

46. This declaration explains the conclusions I have formed based on my analysis. These conclusions are summarized in the table below:

Ground	Basis	Relied-On References	Claim(s)
1	Obviousness	Sapuntzakis and Holtzman	1-13
2	Obviousness	Sapuntzakis and Federwisch	14-27
3	Obviousness	Birse and Holtzman	1-13

Ground	Basis	Relied-On References	Claim(s)
4	Obviousness	Birse and Federwisch	14-27

VII. TECHNOLOGY OVERVIEW

47. In this section, I provide an overview of the background art as understood by a POSITA during the relevant timeframe.

A. General Computer Operations

48. Computers are hardware and software devices that execute instructions. The hardware often includes a central processing unit ("CPU") or other microprocessors, typically called "processors," volatile memory (*e.g.*, RAM (random-access memory), or DRAM (dynamic random-access memory)), non-volatile (persistent) storage (*e.g.*, a hard disk or flash drive), and input/output ("I/O") devices (*e.g.*, network interface card, mouse, keyboard, display). RAM, hard disks, and flash drives are considered, generally, "storage" or "memory" devices because they can store and hold bits of information (whether persistently or not). Computers can have multiple storage devices. Processors are coupled to main memory using a memory bus and coupled to peripherals such as hard disks and network interface cards using one or more I/O busses.

49. Memory and storage devices have considerable differences in their speeds, capacities, and costs. Non-volatile storage devices such as hard disks have

the largest capacity, are least expensive per gigabyte, and are slowest (typically operating at millisecond speeds). Volatile DRAM devices are smaller, more expensive, and faster than hard disks (typically operating at microsecond speeds). A CPU's own memory caches and operations are the smallest, most expensive, and fastest (typically operating at nanosecond speeds). Overall, CPUs are about 1,000× faster than DRAM, which in turn is about 1,000× faster than hard disks.

50. Computer programs are often written in a high-level human-readable language (*e.g.*, C, C++, Java, Perl, PHP, Python), then translated using a compiler or script processor to machine instructions understood by the CPU (*e.g.*, Intel or AMD processor). Machine instructions—generally called "software"—are stored in files on persistent media (*e.g.*, HDD or ROM), loaded into DRAM, and then loaded into the CPU where they can be executed. All modern computers operate in this manner, regardless of their purpose: a small laptop, a server in a data center, a mainframe computer, a personal workstation, a proxy device, a gateway, a firewall, a router, an appliance, a virtual machine, a handheld or mobile device, a smart card, a SD card, etc. Computer systems may run multiple software programs, from background servers and daemons to user applications.

51. Users (including administrators) can login to one or more computers (*e.g.*, desktop, laptop, server) and execute one or more applications (*e.g.*, word processing, web browsing, backup tool). Some computers offer external interfaces

and APIs to communicate with certain devices and especially peripherals. For example, both a USB storage device as well as a printer use a well-defined "protocol" to allow users to communicate with them through the main computer.

1. Networking Overview

52. A Local Area Network ("LAN") is typically a network at a local site where network latencies are short and physical distances between computers are small. A LAN and its computers are often protected from the rest of the Internet using a firewall. EX1011 (Microsoft Computer Dictionary), 276 ("**LAN** *n*. Acronym for **l**ocal **a**rea **n**etwork. A group of computers and other devices dispersed over a relatively limited area and connected by a communications link that enables any device to interact with any other on the network. LANs commonly include microcomputers and shared resources such as laser printers and large hard disks. The devices on a LAN are known as nodes, and the nodes are connected by cables through which messages are transmitted. *See also* baseband network, broadband network, bus network, collision detection, communications protocol, contention, CSMA/CD, network, ring network, star network, token bus network, token passing, token ring network. *Compare* wide area network.").

53. Another form of a local area network is one that operates inside a single computer. A "localhost" is "[t]he name that is used to represent the same computer on which a TCP/IP message originates. An IP packet sent to localhost has the IP

address 127.0.0.1 and does not actually go out to the Internet. *See also* IP address, packet (definition 1), TCP/IP." EX1011 (Microsoft Computer Dictionary), 287. Often, when users/processes want to communicate with other users/processes, using TCP/IP communications on the same machine, they can use the localhost address: it is considered a trusted, high-speed internal network.

54. Conversely, a Wide-Area Network ("WAN") refers to computers across the entire Internet, which are often distant from each other, exhibit longer latencies, and afford little protection. EX1011 (Microsoft Computer Dictionary), 507 ("**wide area network** *n*. A communications network that connects geographically separated areas. *Acronym*: WAN.").

55. A firewall typically intercepts all traffic between computers in a LAN and the WAN, monitors and controls access, and can filter out undesired traffic. Often, LAN computers are far more trusted than WAN computers; therefore, a firewall can be said to separate the trusted network (LAN or localhost) from the untrusted network (WAN). EX1011 (Microsoft Computer Dictionary), 197 ("**firewall** *n*. A security system intended to protect an organization's network against external threats, such as hackers, coming from another network, such as the Internet. A firewall prevents computers in the organization's network from communicating directly with computers external to the network and vice versa. Instead, all communication is routed through a proxy server outside of the organization's

network, and the proxy server decides whether it is safe to let a particular message or file pass through to the organization's network.").

56. A server is a computer or software that provides a service to other computers, called clients. There are many servers on the Internet, for example, Web servers and Mail servers. EX1011 (Microsoft Computer Dictionary), 430 ("**server** *n*. 1. On a local area network (LAN), a computer running administrative software that controls access to the network and its resources, such as printers and disk drives, and provides resources to computers functioning as workstations on the network. 2. On the Internet or other network, a computer or program that responds to commands from a client. For example, a file server may contain an archive of data or program files; when a client submits a request for a file, the server transfers a copy of the file to the client. *See also* client/server architecture. *Compare* client (definition 3).").

57. A client is typically an end-point computer or software, sometimes used by a user, that communicates across LAN and WAN networks; a client may contact servers to request some service. A user's desktop, laptop, and handheld device are examples of clients. (Note that a computer can be both a client and a server.) EX1011 (Microsoft Computer Dictionary), 92 ("**client** *n*. 1. In object-oriented programming, a member of a class (group) that uses the services of another class to which it is not related. *See also* inheritance (definition 1). 2. A process, such as a program or task, that requests a service provided by another program—for example,

a word processor that calls on a sort routine built into another program. The client process uses the requested service without having to 'know' any working details about the other program or the service itself. *Compare* child (definition 1), descendant (definition 2). 3. On a local area network or the Internet, a computer that accesses shared network resources provided by another computer (called a server). *See also* client/server architecture, server.").

58. A client/server architecture describes how client computers interact with server computers. EX1011 (Microsoft Computer Dictionary), 92 ("**client/server architecture** *n*. An arrangement used on local area networks that makes use of distributed intelligence to treat both the server and the individual workstations as intelligent, programmable devices, thus exploiting the full computing power of each. This is done by splitting the processing of an application between two distinct components: a 'front-end' client and a 'back-end' server. The client component is a complete, stand-alone personal computer (not a 'dumb' terminal), and it offers the user its full range of power and features for running applications. The server component can be a personal computer, a minicomputer, or a mainframe that provides the traditional strengths offered by minicomputers and mainframes in a time-sharing environment: data management, information sharing between clients, and sophisticated network administration and security features. The client and server machines work together to accomplish the processing of the

application being used. Not only does this increase the processing power available over older architectures but it also uses that power more efficiently. The client portion of the application is typically optimized for user interaction, whereas the server portion provides the centralized, multiuser functionality. *See also* distributed intelligence.”).

B. Data Storage Devices and File Systems

59. As mentioned in §VII.A above, computers often include two types of storage or memory devices. The first type is a volatile memory device, called Random Access Memory (“RAM”), that is often used to hold executing programs and their data. EX1011 (Microsoft Computer Dictionary), p. 395 (“RAM”).

60. The second type is a non-volatile, or persistent storage device, such as a hard disk drive (“HDD”). HDDs are used to store programs and data persistently. EX1011 (Microsoft Computer Dictionary), p. 226 (“hard disk”).

61. HDDs are electro-mechanical devices with moving parts. Computers can also have other types of non-volatile storage devices that have no moving parts, also called “solid-state devices,” such as flash memories. EX1011 (Microsoft Computer Dictionary), p. 199 (“flash memory”).

62. Moreover, any single device (*e.g.*, a hard disk) can be logically broken down into several units called “partitions.” EX1011 (Microsoft Computer Dictionary), p. 152 (“**disk partition** *n.* A logical compartment on a physical disk

drive. A single disk might have two or more logical disk partitions, each of which would be referenced with a different disk drive name. Multiple partitions are divided into a primary (boot) partition and one or more extended partitions.”).

63. Above the storage media often lives a logical software layer called a “file system.” File systems provide a convenient abstraction to users and applications of files, folders or directories, and naming. EX1011 (Microsoft Computer Dictionary), p. 196 (“**file system** *n.* In an operating system, the overall structure in which files are named, stored, and organized. A file system consists of files, directories, or folders, and the information needed to locate and access these items. The term can also refer to the portion of an operating system that translates requests for file operations from an application program into low-level, sector-oriented tasks that can be understood by the drivers controlling the disk drives. *See also* driver.”).

1. Files and File Systems

64. A file is a collection of data and attributes, or meta-data, about the file, including the file’s name and extension. EX1011 (Microsoft Computer Dictionary), p. 194 (“**file** *n.* A complete, named collection of information, such as a program, a set of data used by a program, or a user-created document. A file is the basic unit of storage that enables a computer to distinguish one set of information from another. A file is the ‘glue’ that binds a conglomeration of instructions, numbers, words, or

images into a coherent unit that a user can retrieve, change, delete, save, or send to an output device.”).

65. File attributes are also called file meta-data and include security attributes, the name (title), size of a file, timestamps, and more. EX1011 (Microsoft Computer Dictionary), p. 305 (“**meta data** or **metadata** *n.* 1. Data about data. For example, the title, subject, author, and size of a file constitute metadata about the file. *See also* data dictionary, repository.”), p. 194 (“**file attribute** *n.* A restrictive label attached to a file that describes and regulates its use—for example, hidden, system, read-only, archive, and so forth. In MS-DOS, this information is stored as part of the file’s directory entry.”).

66. A file’s name is an important attribute of a file and conveys a lot of useful information about the file’s meaning and purpose. EX1011 (Microsoft Computer Dictionary), p. 195 (“**filename** *n.* The set of letters, numbers, and allowable symbols assigned to a file to distinguish it from all other files in a particular directory on a disk. A file name is the label under which a computer user saves and requests a block of information. Both programs and data have file names and often extensions that further identify the type or purpose of the file. Naming conventions, such as maximum length and allowable characters of a file name, vary from one operating system to another. *See also* directory, path (definition 5).”).

67. A file's path or path-name conveys the location of a file in the name hierarchy and even which file server the file may reside on (*e.g.*, the local C:\ drive or a network drive like N:\). EX1011 (Microsoft Computer Dictionary), p. 356 ("**path** *n.* […] 5. In file storage, the route followed by the operating system through the directories in finding, sorting, and retrieving files on a disk."), *id.* ("**pathname** *n.* In a hierarchical filing system, a listing of the directories or folders that lead from the current directory to a file. *Also called:* directory path.").

68. Moreover, a file's extension conveys vital information about the file's type and hence its meaning and purpose. For example, a file ending with "xls" is known to be a Microsoft Excel spreadsheet, files with "doc" extensions are Microsoft Word files, and those ending with "html" are Web server hyper-text markup language files. EX1011 (Microsoft Computer Dictionary), p. 194 ("**file extension** *n.* *See* extension (definition 1)."), p. 195 ("**filename extension** *n. See* extension (definition 1)."), p. 186 ("**extension** *n.* 1. A set of characters added to a filename that serves to extend or clarify its meaning or to identify a file as a member of a category. An extension may be assigned by the user or by a program, as, for example, .com or .exe for executable programs that MS-DOS can load and run.").

69. The (usually non-volatile) storage media that holds the actual bits of file data is often called the "secondary storage" or the "backing storage" (because it is in the "back end"); this is different than "backup storage" that's often used to

backup files for disaster recovery purposes. A "primary storage" is "Random access memory (RAM); the main general-purpose storage region to which the microprocessor has direct access. A computer's other storage options, such as disks and tape, are called *secondary storage* or (sometimes) *backing storage*." EX1011 (Microsoft Computer Dictionary), pp. 378-379. A "secondary storage" is "Any data storage medium other than a computer's random access memory (RAM)-typically tape or disk. *Compare* primary storage." EX1011 (Microsoft Computer Dictionary), p. 425.

70. Above the storage media often lives a logical software layer called a "file system." File systems provide a convenient abstraction to users and applications of files, folders or directories, and naming. EX1011 (Microsoft Computer Dictionary), p. 196 ("**file system** *n.* In an operating system, the overall structure in which files are named, stored, and organized. A file system consists of files, directories, and the information needed to locate and access these items. The term can also refer to the portion of an operating system that translates requests for file operations from an application program into low-level, sector-oriented tasks that can be understood by the drivers controlling the disk drives. *See also* driver.").

71. The "FAT file system" is "[t]he system used by MS-DOS to organize and manage files. The FAT (file allocation table) is a data structure that MS-DOS creates on the disk when the disk is formatted. When MS-DOS stores a file on a

formatted disk, the operating system places information about the stored file in the FAT so that MS-DOS can retrieve the file later when requested. The FAT is the only file system MS-DOS can use; OS/2, Windows NT, and Windows 95 operating systems can use the FAT file system in addition to their own file systems (HPFS, NTFS, and VFAT, respectively). *See also* file allocation table, HPFS, NTFS, OS/2, VFAT, Windows 95, Windows NT."). EX1011 (Microsoft Computer Dictionary), pp. 189-190.

72. Finally, before any storage media or partition can be used to access files and folders, it must be formatted. To "format" is "2. To prepare a disk for use by organizing its storage space into a collection of data 'compartments,' each of which can be located by the operating system so that data can be sorted and retrieved. When a previously used disk is formatted, any preexisting information on it is lost." EX1011 (Microsoft Computer Dictionary), p. 204 ("Format" definition 2).

C. Networked and Distributed Storage Systems

73. As discussed in §VII.B above, storage devices provide raw storage to computers. One of the most popular storage device protocols is SCSI, which is an "[a]cronym for **S**mall **C**omputer **S**ystem **I**nterface) a standard high-speed parallel interface defined by the X3T9.2 committee of the American National Standards Institute (ANSI). A SCSI interface is used to connect microcomputers to SCSI peripheral devices, such as 1nany hard disks and printers, and to other computers

and local area networks. *Compare* ESDI, IDE." EX1011 (Microsoft Computer Dictionary), p. 423.

74. A "SCSI network" is "[a] set of devices on a SCSI bus, which acts like a local area network. *See also* SCSI." EX1011 (Microsoft Computer Dictionary), p. 424.

75. Indeed, the SCSI protocol was fairly similar to other network protocols, that the IETF began standardizing it in the early 2000s. iSCSI is "a transport protocol for Internet Small Computer Systems Interface (iSCSI) that works on top of TCP. The iSCSI protocol aims to be fully compliant with the standardized SCSI architecture model. … As system interconnects move from the classical bus structure to a network structure, SCSI has to be mapped to network transport protocols. IP networks now meet the performance requirements of fast system interconnects and as such are good candidates to 'carry' SCSI." EX1017 (RFC 3720), p. 1.

76. File systems can appear as local volumes (*e.g.*, the C:\ drive in Windows), but they can also be networked, where the physical storage resides on a remote file server. Network file systems allow files and volumes to be shared and accessible from multiple networked computers. EX1011 (Microsoft Computer Dictionary), p. 330 ("**Network File System** *n.* *See* NFS."), p. 328 ("**NFS** *n.* Acronym for **N**etwork **F**ile **S**ystem. A distributed file system developed by Sun Microsystems, Inc. that allows users of Windows NT and UNIX workstations to

access remote files and directories on a network as if they were local. *Acronym*: NFS.”).

77. Even storage media can be networked into a large cluster of devices, that can export virtual block devices to individual hosts, called a “storage area network” (“SAN”).

78. When the amount of data to access grew large enough, and especially if the data had to be concurrently accessed from multiple users/applications, storage researchers and companies leveraged the principles of distributed systems to improve the performance, resiliency, and fault-tolerance of the storage system.

79. A single computer’s performance is limited by the physical maximum speeds of its network, processor, memory, and I/O devices. A distributed system, however, improves overall performance by using multiple computers concurrently to process many requests, thus increasing the aggerate throughput manyfold.

80. A single computer also represents a single point of failure: if that one computer is down, off, or broken into, then the service it offers is unavailable. A distributed system, conversely, improves resiliency and security by ensuring that the failure of any one (or even a few) components do not affect the overall service provided. This principle is called “fault tolerance.” EX1011 (Microsoft Computer Dictionary), p. 190 (“**fault tolerance** *n.* The ability [] to respond to a catastrophic event or fault, such as a power outage or a hardware failure, in a way that ensures

that no data is lost and any work in progress is not corrupted. [] In a fault-tolerant network, the system has the ability either to continue the system's operation without loss of data or to shut the system down and restart it, recovering all processing that was in progress when the fault occurred.").

81. NFS is but one example of a networked or distributed file system. Many complex software systems have been designed to operate in a distributed manner, where they are more resilient to failures and scale better than software running on a single node.

82. Generally, a "distributed network" is "[a] network in which processing, storage, and other functions are handled by separate units (nodes) rather than by a single main computer." EX1011 (Microsoft Computer Dictionary), p. 154.

83. The "Distributed Computing Environment" is "[a] set of standards from the Open Group (formerly the Open Software Foundation) for development of distributed applications that can operate on more than one platform. *Acronym*: DCE. *See also* distributed processing." EX1011 (Microsoft Computer Dictionary), pp. 153-154.

84. "[D]istributed processing" is "[a] form of information processing in which work is performed by separate computers linked through a communications network. Distributed processing is usually categorized as either plain distributed processing or true distributed processing. Plain distributed processing shares the

workload among computers that can communicate with one another. True distributed processing has separate computers perform different tasks in such a way that their combined work can contribute to a larger goal. The latter type of processing requires a highly structured environment that allows hardware and software to communicate, share resources, and exchange information freely." EX1011 (Microsoft Computer Dictionary), p. 154.

85. Finally, a "distributed file system" is "[a] file management system in which files may be located on multiple computers connected over a local or wide area network." EX1011 (Microsoft Computer Dictionary), p. 154.

D. Caching Technologies

86. Computers store the 0 and 1 values of bits in memory devices. Memory devices can be volatile (lose data when device power is off) or non-volatile (data is not lost when power is off). There are many storage media in existence: tapes, hard disks, floppies, Flash technologies (*e.g.*, SSDs), optical (*e.g.*, CD-ROM and DVDs), DRAM (*e.g.*, system memory), CPU registers, and more. All these devices can be placed on a specific point or range in a 3-dimensional continuum comprising their (a) capacity, (b) performance, and (c) price. Almost universally, the faster performing devices (*e.g.*, DRAM) are smaller and cost more per unit storage, whereas the larger-capacity devices (*e.g.*, HDD) are slower and cheaper per gigabyte of storage capacity. The difference in speed/cost/capacity among devices is very

significant and ranges along at least 9 orders of magnitude—or a billion times bigger, faster, or costlier. *See* §VII.A above.

87. Therefore, "[t]he wide variety of storage systems in a computer system can be organized in a hierarchy (Figure 2.6) according to their speed and their cost. The higher levels are expensive, but are fast. As we move down the hierarchy, the cost per bit decreases, whereas the access time increases. This tradeoff is reasonable; if a given storage system were both faster and less expensive than another — other properties being the same — then there would be no reason to use the slower, more expensive memory." EX1012 (Silberschatz), 42, Fig. 2.6.

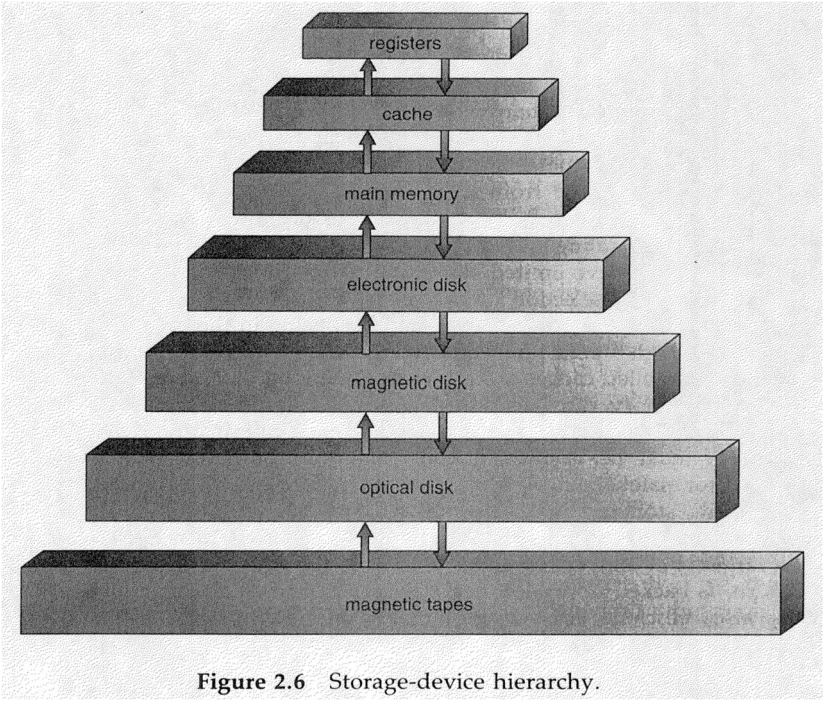


Figure 2.6 Storage-device hierarchy.

88. Managing all these factors is challenging: cost, speed, access speed/latency, size, and volatility. "The design of a complete memory system attempts to balance all these factors: It uses only as much expensive memory as

necessary, while providing as much inexpensive, nonvolatile, memory as possible. Caches can be installed to ameliorate performance differences where there is a large access-time or transfer-rate disparity between two components". EX1012 (Silberschatz), 42-43.

89. "*Caching* is an important principle of computer systems, in both hardware and software. Information is normally kept in some storage system (such as main memory). As it is used, it is copied into faster storage system, the cache, on a temporary basis. When we need a particular piece of information, we first check whether it is in the cache. If it is, we used the information directly from the cache; if it is not, we use the information from the main storage system, putting a copy in the cache under the assumption that there is a high probability that it will be needed again". EX1012 (Silberschatz), 43.

90. A "cache" is "[a] special memory subsystem in which frequently used data values are duplicated for quick access. A memory cache stores the contents of frequently accessed RAM locations and the addresses where these data items are stored. When the processor references an address in memory, the cache checks to see whether it holds that address. If it does hold the address, the data is returned to the processor; if it does not, a regular memory access occurs. A cache is useful when RAM accesses are slow compared with the microprocessor speed, because cache

memory is always faster than main RAM memory. *See also* disk cache, wait state." EX1011 (Microsoft Computer Dictionary), 72.

91. Caches can exist at many levels. A "CPU cache" is "[a] section of fast memory linking the central processing unit (CPU) and main memory that temporarily stores data and instructions the CPU needs to execute upcoming commands and programs. Considerably faster than main memory, the CPU cache contains data that is transferred in blocks, thereby speeding execution. The system anticipates the data it will need through algorithms. *Also called* cache memory, memory cache. *See also* cache, central processing unit, VCACHE." EX1011 (Microsoft Computer Dictionary), 122. CPUs internally can have 2 or more caches, marked "level 1" or L1, "level 2" or L2, etc. *See also* EX1011 (Microsoft Computer Dictionary), 276 ("L1 cache" and "L2 cache" definitions).

92. A "disk cache" "[a] portion of a computer's random access memory (RAM) set aside for temporarily holding information read from disk. A disk cache does not hold entire files, as does a RAM disk (a portion of memory that acts as if it were a disk drive). Instead, a disk cache is used to hold information that either has recently been requested from disk or has previously been written to disk. If the required information remains in a disk cache, access time is considerably faster than if the program must wait for the disk drive mechanism to fetch the information from

disk. *See also* cache. *Compare* disk buffer." EX1011 (Microsoft Computer Dictionary), 151.

93. A "RAM cache" is "[c]ache memory that is used by the system to store and retrieve data from the RAM. Frequently accessed segments of data may be stored in the cache for quicker access compared with secondary storage devices such as disks. *See also* cache, RAM." EX1011 (Microsoft Computer Dictionary), 395.

94. In any cache hierarchy, typically the cache (above) is faster and smaller than the backend location (below) from where the data was retrieved. Therefore, caches must be managed appropriately. If a data item is found in a cache it is called a "cache hit"—otherwise it is a "cache miss" that requires retrieving the backend data and copying it into the cache. A "hit" is "1. A successful retrieval of data from a cache rather than from the slower hard disk or RAM. *See also* cache, hard disk, RAM. 2. A successful retrieval of a record matching a query in a database. *See also* query (definition 1), record[]. 3. Retrieval of a document, such as a home page, from a Web site." EX1011 (Microsoft Computer Dictionary), pp. 234-235.

95. Cached data is typically stored temporarily. At some point a cache space fills up, and the system must determine which items to "evict"; often the oldest, least used items are evicted first. "Since caches have limited size, *cache management* is an important design problem. Careful selection of the cache size and of a replacement policy can result in 80 to 99 percent of all accesses being in the

cache, resulting in extremely high performance. Various replacement algorithms for software-controlled caches are discussed in Chapter 9." EX1012 (Silberschatz), 44.

96. Such caching techniques are particularly effective when considering users' work habits and workload patterns. Studies have shown that users and applications often operate on a small set of data files (called a "working set") within a narrow time period. For example, while drafting this very declaration, I am editing the same Word document repeatedly over a period of several days and weeks. Thus, my OS (and any OS prior to 1998) recognizes this access pattern and keeps this Word file cached in DRAM for long time periods. Conversely, it makes less sense for the OS to waste precious DRAM to cache documents I've not worked on for months. However, if, for any reason, I try to access a document I've not opened in a long time, and the document is not already cached in DRAM, then indeed my OS will have to incur a longer latency—and I will have to wait for however long it takes to retrieve the older document from the slower HDD (and cache it into DRAM).

97. Cache management considers when to "admit" data into the cache from backend sources and when to "evict" older data in the cache to make room for new items. One of the most popular and successful cache eviction algorithms is Least-Recently-Used ("LRU"), a heuristic stating that items used more recently are more likely to be used in the near future, and therefore should be kept—while lesser used items should be evicted. LRU uses various thresholds to determine when and what

to evict, based on space available in the cache, the desire to make extra room in the cache for new items, and/or the length of time that items have been in the cache and when were they last used. "LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses that page that has not been used for the longest period of time." *See* EX1012 (Silberschatz), 320; *see also, generally*, *id.*, at 319-325.

98. Admitting an item from a backend into a cache, or retrieving it from the cache itself, can be seen as a "get" (or read) operation. Conversely, an application inserting an item into a cache, is also called "put" (or write) operation. When new items are inserted into a cache by an application, they are said to be "dirty" because their content is newer than the corresponding backend item. Such dirty items must be "flushed" or written to the backend before they are removed from the cache. A "dirty bit" is "[a] bit used to mark modified data in a cache so that the modifications may be carried over to primary memory. *See also* bit, cache." EX1011 (Microsoft Computer Dictionary), 149.

99. To "flush" is "[t]o clear a portion of memory. For example, to flush a disk file buffer is to save its contents on disk and then clear the buffer for filling again." EX1011 (Microsoft Computer Dictionary), 202 (definition 2). In some caches, items in the cache could become "stale" if the corresponding backend item

could have been updated: in that case, some caches attempt to revalidate any cached item before returning it to an application, and as needed remove stale items.

100. Caching is applicable anywhere there is room in one memory device to access data much faster than its source location. Other than RAM-HDD caches, one can cache items on a local storage device that would otherwise take even longer to retrieve than the HDD's access speeds (*e.g.*, network storage, distant servers). A "proxy server" is "[a] firewall component that manages Internet traffic to and from a local area network (LAN) and can provide other features, such as document caching and access control. A proxy server can improve performance by supplying frequently requested data, such as a popular Web page, and can filter and discard requests that the owner does not consider appropriate, such as requests for unauthorized access to proprietary files. *See also* firewall." EX1011 (Microsoft Computer Dictionary), 387.

101. There are dozens of cache-management algorithms in existence, some decades old. Silberschatz describes several algorithms used to manage memory resources in an OS. *See* EX1012 (Silberschatz), 315-325 ("9.5 Page-Replacement Algorithms", describing techniques such as FIFO, LRU, LFU, MFU, and more), 315 ("There are many different page-replacement algorithms. Probably every operating system has its own unique replacement scheme. How do we select a particular

replacement algorithm? In general, we want the one with the lowest *page-fault rate*.").

1. Least-Recently Used (LRU)

102. One of the oldest, most effective, and best-known algorithms is the Least-Recently-Used (LRU) algorithm. LRU leverages the aforementioned property that users are more likely to work on data files that were in active use in the recent past. LRU orders all cached items by the last time they were accessed. When the cache needs to make room for new items, LRU evicts enough items starting from the least-recently (*e.g.*, oldest) items first. *See* EX1012 (Silberschatz), 319-320 ("**9.5.3 LRU Algorithm** [] If the optimal algorithm is not feasible, perhaps an approximation to the optimal algorithm is possible. … If we use the recent past as an approximation of the near future, then we will replace the page that *has not been used* for the longest period of time (Figure 9.11). This approach is the *least recently used (LRU)* algorithm. LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses that page that has not been used for the longest period of time.").

2. Least-Frequently Used (LFU)

103. Another popular algorithm is Least-Frequently-Used ("LFU"). LFU counts not the recency but the frequency of access of cached items. LFU evicts items that have the lowest frequency of use first. EX1012 (Silberschatz), 325 ("**LFU**

Algorithm: The *least frequently used (LFU)* page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count.”).

104. Neither LRU nor LFU are perfect. Both work well for certain access patterns but may make poor decisions for other access patterns, possibly evicting items that are needed shortly after eviction. For that reason, researchers have developed cache-management techniques that consider other properties and thresholds: the size of the cached object, object priorities, who are the users who requested the item, inter-relationship among items (*e.g.*, items A and B are needed together), and numerous permutations and combinations (*e.g.*, LRU+LFU) over many years, starting in the 1970s. In sum, all cache management techniques attempt to prioritize cached data—which data should be evicted (because it is less important) and which should be kept (because it is deemed more important).

105. While these techniques are described here in the context of cache management, they are generally applicable anywhere resources are managed. When resources are managed, resource consumers are often prioritized. Such priorities are vital to determine which resources to move elsewhere, while minimizing disruption to operations; it is obvious that, to minimize disruption, resource consumers with the lowest priority should be removed (or discarded) first. For example, one can use LRU, LFU, etc. to prioritize resource users and decide which applications or VMs

to migrate from one host (or data center) to another, which network packets to transmit first, which applications to schedule to run first, which web pages to cache locally, etc.

106. And once again, poor priorities and "predictions" can result in less favorable outcomes. For example, a VM that wasn't deemed a high priority before could be offloaded to a distant data center with fewer resources; but if that VM (or the applications it runs) suddenly becomes more important (*e.g.*, due to a spike of activity), then that VM and its application will suffer poor performance, whereas if that VM was not migrated in the first place, it would have performed much better now.

E. Copy-on-Write and Snapshots

107. Copy-on-Write (CoW) has been used in volatile memory systems inside most operating systems. But CoW can be used elsewhere. Most responsible users would make periodic backups of their data files, often on a daily schedule. Backups are therefore a point-in-time copy of users' data files. Network Appliance (NetApp), a company producing storage appliances, popularized the idea of using CoW inside storage systems. Netapp called these point-in-time copies "snapshots." Netapp storage appliances (which I've used and maintained for nearly 20 years, *see* §II above) can periodically take read-only snapshots of the storage appliance, thus producing a point-in-time read-only image of the storage system. Snapshots can be

taken more frequently (*e.g.*, every hour or even few minutes). Snapshots also save on disk space, because data blocks files/sectors that have no changed blocks between snapshots, are shared across multiple snapshots; this is a form of data "de-duplication" that can save a lot of space compared to, say, taking full (and even incremental file-based) backups periodically.

108. Netapp's appliances perform CoW and manage snapshots at the (disk) block layer, but they include a full-fledged file system layer on top of the block layer. That file system can be accessible over the network using NFS (for Unix hosts) or CIFS/Samba (for Windows hosts). *See* §VII.B.1 above. Netapp's CoW engine/driver therefore resides between the file system layer and the actual storage devices.

109. Netapp designed a file system optimized for snapshots and network access, using CoW. "WAFL (Write Anywhere File Layout) … is a file system designed specifically to work in an NFS appliance. The primary focus is on the algorithms and data structures that WAFL uses to implement Snapshots™, which are read-only clones of the active file system. WAFL uses a copy-on-write technique to minimize the disk space that Snapshots consume." EX1013 ("Hitz"), p. 4.

110. Snapshots can be created and deleted as needed, offering space saving and improved reliability. "WAFL's primary distinguishing characteristic is Snapshots, which are read-only copies of the entire file system. WAFL creates and

deletes Snapshots automatically at prescheduled times, and it keeps up to 20 Snapshots on-line at once to provide easy access to old versions of files. Snapshots use a copy-on-write technique to avoid duplicating disk blocks that are the same in a Snapshot as in the active file system. Only when blocks in the active file system are modified or removed do Snapshots containing those blocks begin to consume disk space. Users can access Snapshots through NFS to recover files that they have accidentally changed or removed, and system administrators can use Snapshots to create backups safely from a running system. In addition, WAFL uses Snapshots internally so that it can restart quickly even after an unclean system shutdown." EX1013 ("Hitz"), p. 6.

111. "The FAServer has commands to let system administrators create and delete Snapshots, but it creates and deletes most Snapshots automatically." EX1013 ("Hitz"), p. 7.

112. Netapp maintains meta-data inside files as well as allocation bitmaps. "**3.2. Meta-Data Lives in Files** [] Like Episode, WAFL stores meta-data in files. WAFL's three meta-data files are the inode file, which contains the inodes for the file system, the block-map file, which identifies free blocks, and the inode-map file, which identifies free inodes. The term 'map' is used instead of 'bit map' because these files use more than one bit for each entry. The block-map file's format is described in detail below." EX1013 ("Hitz"), p. 8.

113. Indeed, files can transparently map to multiple data and meta-data blocks in WAFL. When a file is modified, any of its blocks that need to change are copied first, using standard CoW techniques. "Figure 2 … shows that files are made up of individual blocks and that large files have additional layers of indirection between the inode and the actual data blocks. In order for WAFL to boot, it must be able to find the root of this tree, so the one exception to WAFL's write-anywhere rule is that the block containing the root inode must live at a fixed location on disk where WAFL can find it." EX1013 ("Hitz"), pp. 9-10, Figure 2 (reproduced below).

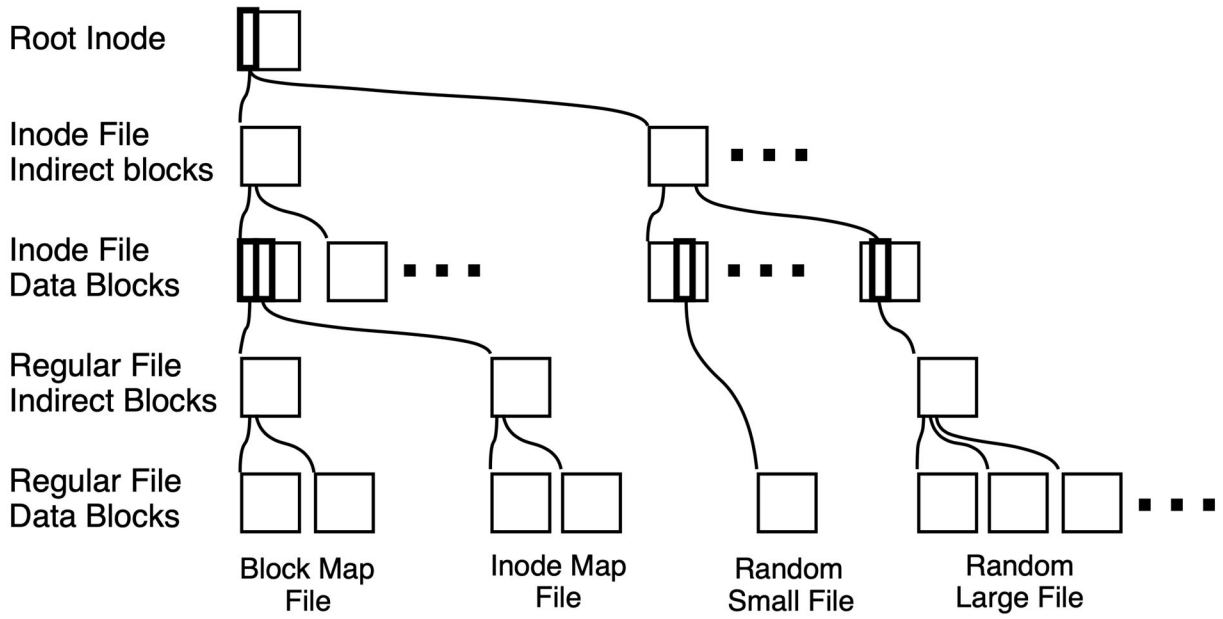


Figure 2
A more detailed view of WAFL's tree of blocks.

114. Netapp details how snapshot works in the presence of file modifications. "**3.4. Snapshots** … Understanding that the WAFL file system is a tree of blocks rooted by the root inode is the key to understanding Snapshots. To

create a virtual copy of this tree of blocks, WAFL simply duplicates the root inode. Figure 3 shows how this works. Figure 3(a) is a simplified diagram of the file system in Figure 2 … Figure 3(b) shows how WAFL creates a new Snapshot by making a duplicate copy of the root inode. … Figure 3(c) shows what happens when a user modifies data block D. WAFL writes the new data to block D' on disk, and changes the active file system to point to the new block. The Snapshot still references the original block D which is unmodified on disk. Over time, as files in the active file system are modified or deleted, the Snapshot references more and more blocks that are no longer used in the active file system. The rate at which files change determines how long Snapshots can be kept on line before they consume an unacceptable amount of disk space." EX1013 ("Hitz"), pp. 10-11, Figure 3 (reproduced below).

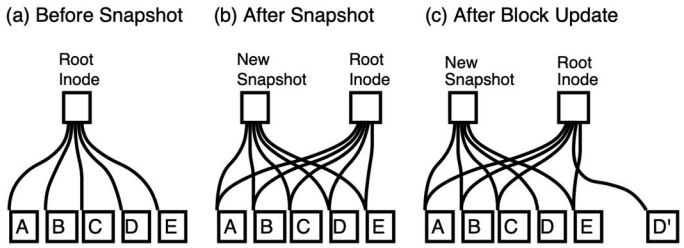


Figure 3
WAFL creates a Snapshot by duplicating the root inode that describes the inode file. WAFL avoids changing blocks in a Snapshot by writing new data to new locations on disk.

115. Initially, Netapp's snapshots were all read-only, with modifications allowed only at the top of the snapshot stack. This created a linear set of several read-only snapshots followed by the primary writeable storage view, further visible

to applications as a file system. Netapp's next evolution was to permit one to create writeable snapshots, by utilizing CoW of a read-only snapshot.

116. Netapp's developed a "system and method for creating a writable clone of a read-only volume. … [A] base snapshot is generated on a source volume on a source storage system and is duplicated as a read-only base snapshot replica on a target volume on a destination storage system. A copy ('clone') is then substantially instantaneously created from the read-only base snapshot replica, thereby creating a writable clone of a read-only volume." EX1016 ("Fair"), 6:18-41.

117. With writeable clones enabled, Netapp filers could now create a (non-linear) tree of snapshots, some read-only and some cloned and writeable. "[M]ultiple clones 1350 may be spawned from the same replica base snapshot 1320; in this case, information is recorded by setting the same snapshot bit in bit fields in both clone's storage label 30 files 1390." EX1016 ("Fair"), 21:27-31.

118. Now that Netapp had both read-only and writeable snapshots, the next evolution was to transmit a snapshot's data from one networked filer appliance to another, thus creating an identical mirror of the source filer.

119. "Computerized data has become critical to the survival of an enterprise. Companies must have a strategy for recovering their data should a disaster such as a fire destroy the primary data center." EX1014 ("Patterson I"), p. 2.

120. "Mirrors keep backup data on-line and fully synchronized with the primary store, but they do so at a high cost in performance (write latency) and network bandwidth." EX1014 ("Patterson I"), p. 2.

121. "SnapMirror [is] a technology which implements asynchronous mirrors on Network Appliance filers. SnapMirror periodically transfers self-consistent snapshots of the data from a source volume to the destination volume. The mirror is on-line, so disaster recovery can be instantaneous. Users set the update frequency. If the update frequency is high, the mirror will be nearly current with the source and very little data will be lost when disaster strikes. But, by lowering the update frequency, data managers can reduce the performance and network cost of maintaining the mirror at the risk of increased data loss." EX1014 ("Patterson I"), pp. 2-3.

122. Like any operating system and appliance, Netapp utilizes caching heavily. *See also* §VII.D above.

123. "During a write episode, WAFL allocates disk space for all the dirty data in the cache and schedules the required disk I/O." EX1013 ("Hitz"), p. 12. "A file system with an NVRAM write cache does all the same steps, except that it copies modified data into NVRAM instead of waiting for the data to reach disk." EX1013 ("Hitz"), p. 13. "WAFL's technique for keeping Snapshot data self consistent is to mark all the dirty data in the cache as 'IN_SNAPSHOT.'" EX1013 ("Hitz"), p. 17.

124. "WAFL caches inode data in two places: in a special cache of in-core inodes, and in disk buffers belonging to the inode file. When it finishes write allocating a file, WAFL copies the newly updated inode information from the inode cache into the appropriate inode file disk buffer, and clears the IN_SNAPSHOT bit on the in-core inode." EX1013 ("Hitz"), p. 17.

125. "The requested block is then retrieved from disk and stored in a buffer cache of the memory as part of a buffer tree of the file. The buffer tree is an internal representation of blocks for a file stored in the buffer cache and maintained by the file system." EX1016 ("Fair"), 3:1-5.

126. "[P]eriodic updates use the primary volume as a giant write cache and it has long been known that write caches can reduce I/O traffic." EX1014 ("Patterson I"), p. 8.

127. "[W]e used two Network Appliance F760 filers directly connected via Intel GbE. Each utilized an Alpha 21164 processor running at 600 MHz, with 1024 MB of RAM plus 32 MB non-volatile write cache." EX1014 ("Patterson I"), p. 12.

128. Any storage system with snapshots would include at least one complete "base" or "root" and one or more incremental (leaf) images, called "snapshots." Because snapshots naturally contain incomplete information, they alone cannot be mounted as a file system. Any blocks not included in one snapshot must be located in a previous point in time snapshot; and those not found there have to be located in

the next previous snapshot; and so on until one gets to the base/root image which is by definition complete. Thus, the process of traversing several snapshots all the way down to the root image would result in merging or unifying one or more disparate snapshots with the root/base image, to produce a merged or unified view of the file system that is consistent and cohesive.

129. EX1015 (Patterson II) describes a system for "[r]esynchronization of a target volume with a source volume" in the presence of snapshots. EX1015 (Patterson II), Title. This is useful to mirror two or more volumes in any direction.

130. Patterson II describes "[a]n improved method and apparatus for quickly and efficiently updating the original source volume and original target volumes after the original source volume has become temporarily unavailable. The original target volume is characterized as a source volume while the original source volume is temporarily unavailable. Transfer lists of different data blocks are generated. Data blocks not originally found on a source are copied to the target. Data blocks included on a target that were not found on the source are removed. By focusing upon specific data blocks, this technique avoids the use of filer overhead and other computational resources that would be expended if the entire volume were recopied." EX1015 (Patterson II), Abstract.

131. Patterson II's Figure 2 (reproduced below) describes this process.

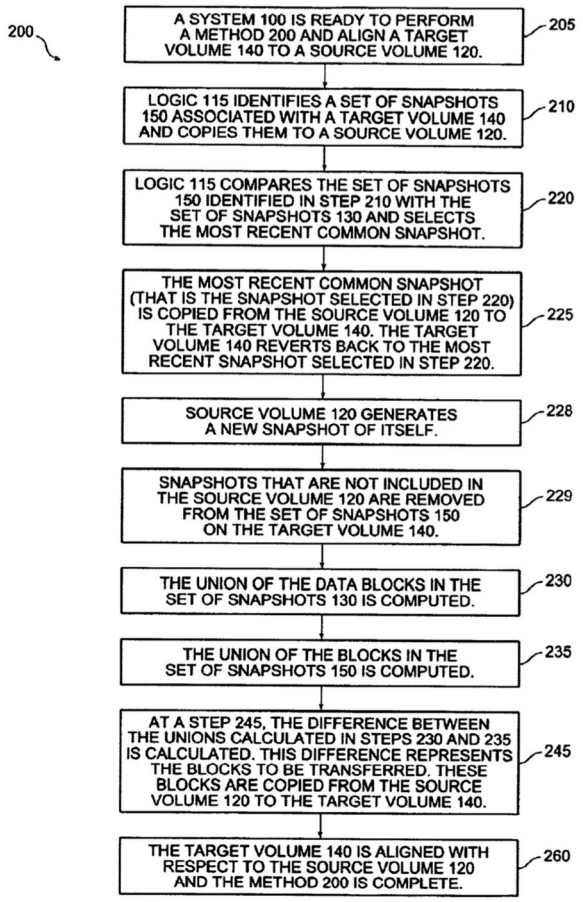


FIG. 2

132. In Figure 2, the system starts by comparing snapshots at the source and destination volumes, to identify a set of differences that have to be transmitted (steps 205-229). The next three steps actually calculate the merged union of the snapshots that need to be transferred, and then transferring those unions.

133. "At a step 230, the union of the data blocks in the set of snapshots 130 is computed. This union will preferably include available and allocated data blocks from any one or more target snapshots 130. This step is preferably performed by the source volume 120. Steps 235 and 245 occur simultaneously with step 230." EX1015 (Patterson II), 5:6-10.

134. "At a step 235, the union of the blocks in the set of snapshots 150 is computed. This union will preferably include available and allocated data blocks from any one or more target snapshots 150. This step is preferably performed by the source volume 120. Step 235 is performed at the same time as steps 230 and 245." EX1015 (Patterson II) 5:11-16.

135. "At a step 245, difference between the unions calculated in steps 230 and 235 is calculated. This difference represents the blocks to be transferred. These blocks are copied from the source volume 120 to the target volume 140. In this way, the target volume 140 becomes synchronized with respect to blocks present on the source volume 120." EX1015 (Patterson II), 5:17-22.

136. Finally, at step 260, the synchronization of the two volumes is complete. A POSITA would easily recognize that the steps Patterson II disclose merge/unify data blocks from multiple snapshots into a set to transmit. This unioning takes place at the block level, because Patterson II's snapshots are performed at the block level (the file system resides logically above). EX1015 (Patterson II), Figure 1 (reproduced below).

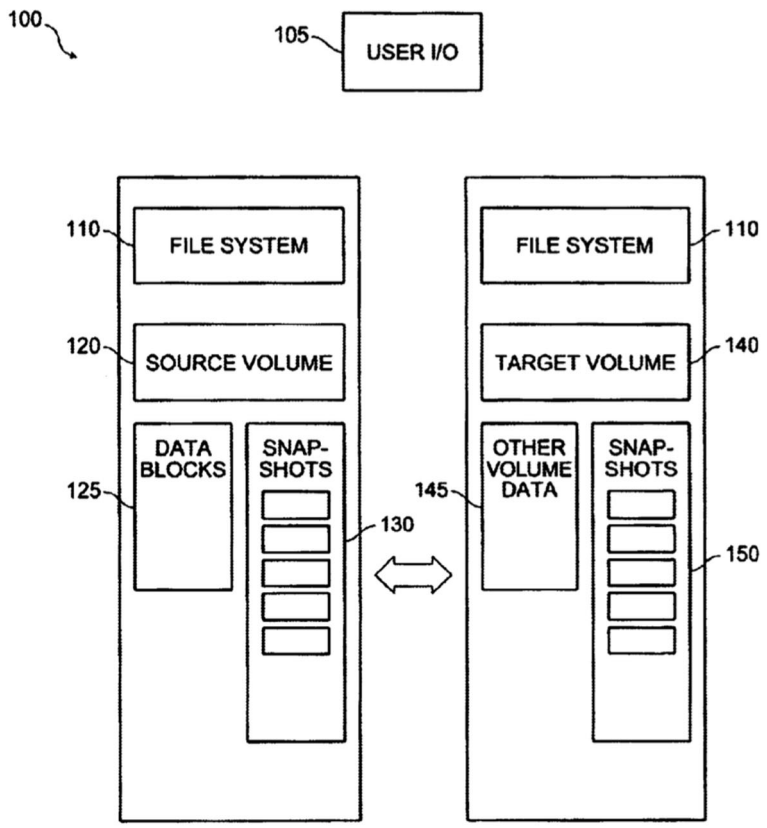


FIG. 1

137. In Patterson II, the software component that performs said unification of snapshots happens to be driven by a file system. "File system software synchronizes the target volume with the source volume. First, the file system software removes snapshots from a target volume if the snapshots are not included in the source volume's snapshot list. Second, the file system software adds the set of data blocks identified above (that is the set of data blocks that are included in the source volume and not included in the target volume) to its memory. Lastly, the file system software adds snapshots to the target volume if the snapshots are included in the source volume's snapshot list and not in the target volume's snapshot list. At

this point, the target volume includes the data blocks that are present on the source volume." EX1015 (Patterson II), 2:22-34.

138. Lastly, in Patterson II, "sources and volumes can be synchronized dynamically, using a WAFL (Write Anywhere File Layout) system using RAID (Redundant Arrays of Independent Disks) architecture. However, various other types of file systems involving redundant copies of data can also be used." EX1015 (Patterson II), 2:47-53. A POSITA would know that RAID is software that operates at the block level. *See* §VII.B above.

F. Indexing

139. Indexing is a general computing technique to organize a catalog for finding information more efficiently (just as a library contains a catalog of all books it has and their shelf location).

140. An "indexed search" is "[a] search for an item of data that uses an index to reduce the amount of time required." EX1011 (Microsoft Computer Dictionary), p. 247.

141. Similarly, users are familiar with many forms of index searches, such as on the WWW. A "Web index" is "[a] Web site intended to enable a user to locate other resources on the Web. The Web index may include a search facility or may merely contain individual hyperlinks to the resources indexed." EX1011 (Microsoft Computer Dictionary), p. 505.

142. There are several forms of indexes and data structures possible. An "index" (definition 1) is "1. A listing of keywords and associated data that point to the location of more comprehensive information, such as files and records on a disk or record keys in a database. 2. In programming, a scalar value that allows direct access into a multi-element data structure such as an array without the need for a sequential search through the collection of elements. *See also* array, element (definition 1), hash, list." EX1011 (Microsoft Computer Dictionary), p. 247.

143. Indexes can be created and managed in various forms, including storage systems, file systems, databases, and more. To "index" (definition 2) is "1. In data storage and retrieval, to create and use a list or table that contains reference information pointing to stored data. 2. In a database, to find data by using keys such as words or field names to locate records. 3. In indexed file storage, to find files stored on disk by using an index of file locations (addresses). 4. In programming and information processing, to locate information stored in a table by adding an offset amount, called the index, to the base address of the table." EX1011 (Microsoft Computer Dictionary), p. 247.

G. Virtualization

144. Among computer scientists, there is an axiom stating that "you can solve almost any problem by adding another level of indirection." Indeed, virtualization is an indirection technique that makes one thing seem like another;

virtualization often presents a physical-world artifact as one that appears as an illusion or "fake." For something to be "virtual", it is "[o]f or pertaining to a device, service, or sensory input that is perceived to be what it is not in actuality, usually as more 'real' or concrete than it actually is." EX1011 (Microsoft Computer Dictionary), p. 497.

145. Indeed, the Microsoft Computer Dictionary includes 35 examples and definitions of virtualization. EX1011 (Microsoft Computer Dictionary), pp. 497-500.

146. In one example, a "virtual machine" is "[s]oftware that mimics the performance of a hardware device, such as a program that allows applications written for an Intel processor to be run on a Motorola chip. *Acronym*: VM." EX1011 (Microsoft Computer Dictionary), p. 498.

147. In another example, "virtual memory" is "[m]emory that appears to an application to be larger and more uniform than it is. Virtual memory may be partially simulated by secondary storage such as a hard disk. Applications access memory through virtual addresses, are translated (mapped) by special hardware and software onto physical addresses. *Acronym*: VM. *Also called* disk memory. *See also* paging, segmentation." EX1011 (Microsoft Computer Dictionary), p. 498.

148. In yet another example, "virtual reality" is "[a] simulated 3-D environment that a user can experience and manipulate as if it were physical. The

user sees the environment on display screens, possibly mounted in a special pair of goggles. Special input devices, such as gloves or suits fitted with motion sensors, detect the user's actions. *Acronym*: VR.". EX1011 (Microsoft Computer Dictionary), p. 499.

149. Virtualization had been applied to storage systems, disk devices, and drivers. A "virtual device driver" is "[s]oftware in Windows 95 that manages a hardware or software system resource. If a resource retains information from one access to the next that affects the way it behaves when accessed (for example, a disk controller with its status information and buffers), a virtual device driver must exist for it." EX1011 (Microsoft Computer Dictionary), p. 498.

150. A "virtual disk" or "RAM disk" is "[s]hort for random access memory disk. A simulated disk drive whose data is actually stored in RAM memory. A special program allows the operating system to read from and write to the simulated device as if it were a disk drive. RAM disks are extremely fast, but they require that system memory be given up for their use. Also, RAM disks usually use volatile memory, so the data stored on them disappears when power is turned off. Many portables offer RAM disks that use battery-backed CMOS RAM to avoid this problem. *See also* CMOS RAM. *Compare* disk cache." EX1011 (Microsoft Computer Dictionary), pp. 395-396, 498.

151. Virtualization is sometimes said to be "logical" representation of physical reality. "Logical" is "[o]f or pertaining to a conceptual piece of equipment or frame of reference, regardless of how it may be realized physically. *Compare* physical." EX1011 (Microsoft Computer Dictionary), p. 288 ("logical" definition 2).

152. A "logical drive" or "logical device" is "[a] device named by the logic of a software system, regardless of its physical relationship to the system. For example, a single floppy disk drive can simultaneously be, to the MS-DOS operating system, both logical drive A and drive B." EX1011 (Microsoft Computer Dictionary), p. 288.

153. Moreover, one can virtualize files and file systems. This was the subject of much of my graduate-level research and became part of my PhD Dissertation. *See* §II above and EX1004 (Zadok Curriculum Vitae).

154. Virtualization, of course, has been applied to networking as well. A "virtual network" is "[a] part of a network that appears to a user to be a network of its own. For example, an Internet service provider can set up multiple domains on a single HTTP server so that each one can be addressed with its company's registered domain name. *See also* domain name, HTTP server (definition 1), ISP." EX1011 (Microsoft Computer Dictionary), p. 499.

155. A "virtual LAN" is "[s]hort for **virtual l**ocal **a**rea **n**etwork. A local area network consisting of groups of hosts that are on physically different segments but that communicate as though they were on the same wire. *See also* LAN." EX1011 (Microsoft Computer Dictionary), p. 498.

156. One of the most famous uses of a (secure) virtual network is a private one. A "virtual private network" ("VPN") is "[a] set of nodes on a public network such as the Internet that communicate among themselves using encryption technology so that their messages are as safe from being intercepted and understood by unauthorized users as if the nodes were connected by private lines. 2. A wide area network formed of permanent virtual circuits (PVCs) on another network, especially a network using technologies such as ATM or frame relay. *Acronym*: VPN. *See also* ATM (definition 1), frame relay, PVC." EX1011 (Microsoft Computer Dictionary), p. 499. A POSITA would have understood definition 2 of VPN to be a form of an overlay network because it is a collection of individual virtual network segments.

VIII. THE '844 PATENT

A. Overview

157. The '844 Patent describes "technology for, among other things root image caching and indexing for block-level distributed application management." EX1001, Abstract. This "involves storing blocks of a root image on a first storage

unit and storing blocks of leaf images on respective second storage units [which] … include additional data blocks not previously contained in the root image and changes made by respective compute nodes to the blocks of the root image." *Id*. A POSITA would have recognized that these leaf images correspond to storage snapshots as I describe in §VII.E above.

158. The '844 Patent then describes common "caching blocks of the root image that have been accessed by at least one compute node." *Id*. Caching was well known and ubiquitous as I describe in §VII.D above.

159. Finally, the '844 Patent describes generating and "receiving indexing results pertaining to the root image from one compute node and providing the results for other compute nodes." *Id*. Again, indexing of data and files was well known in the art, as I described in §VII.F above.

160. The '844 Patent describes a background where, "as the internet has expanded and computers have multiplied, the need for clustered computing such as High Performance Computing (HPC) has increased[,]" and that "[c]lustered computing involves multiple compute nodes, usually a server grid, that work together to achieve a common task." EX1001, 1:31-44.

161. The '844 Patent describes the trend towards global file system management: "[i]n large-scale systems such as this, a trend in software deployment

is to centralize data management on a globally accessible file system with stateless computing nodes." *Id*.

162. That led to the need to manage individual nodes' boot software and/or boot images. "A common example of this is Operating System (OS) software image management, where the compute nodes are activated with the distributed application environment by either diskless booting protocols or remote software installation to local storage. Under this architecture, a boot image is required for each compute node in the cluster." *Id*.

163. OS boot images contain the OS kernel, libraries, applications, and more. "The boot image necessarily contains the kernel; it may additionally contain the application software that is intended to be run on the compute node." *Id*.

164. The '844 Patent explains a concern with managing so many boot images, many of which may be (nearly or) identical. "The primary concern in clustered computing is low cluster bring-up time. The software that provides the boot images for the cluster typically stores a master boot image. It may then either pre-create clones of this master image for each such server, or it may create them 'on the fly.'" EX1001, 1:49-67.

165. Thus, "[c]reating a boot image on the fly involves copying the entire contents of the master image, which are typically in the range of 5-15 GB." *Id*. This

can be slow. "Even with a significant amount of bandwidth by today's standards, this method will result in a large bring-up time." *Id*.

166. One proposed solution is that "[p]re-creating a boot image for each server is advantageous from the point of view of cluster bring-up time." *Id*. But such a solution could waste disk/storage space "since one often does not know in advance how many servers will ever be booted, this scheme may result in wasted disk space." *Id*. Thus, "both [methods] suffer from the same major problem—updating the boot image(s) for the cluster is cumbersome, as it means updating a number of copies of the boot image."

167. The '844 Patent's Background section then discusses how "[i]n a branching store file system, a read-only base image (or 'root' image) of the application environment is created." EX1001, 2:13-24.

168. "The root image is accessible by all compute nodes in the cluster" and "[c]hanges made by a compute node to the root image are stored in a 'leaf' image unique to that compute node" (*i.e.*, a snapshot). *Id*.

169. To make this work, a software driver, component, or filter "operates between the compute nodes and the file system(s), which merges the changes recorded on the leaf images with the root image and delivers the result to the appropriate compute node." *Id*.

170. The benefit is that, "[f]rom the point of view of the compute node, it is running its own unique and cohesive instance of the application environment." *Id*.

171. Nevertheless, "[w]hile this system allows for creation of boot images on the fly without severely diminishing bring-up time, a separate version of the system must be created for each unique operating system because data is stored at the file system level (i.e., on a 'per file basis')." *Id*.

172. Therefore, the '844 Patent argues that "migrating a computing cluster from one operating system to another is much more complicated than simply installing a new root image containing the new OS." *Id*.

173. After discussing and admitting to this known prior art, the '844 Patent describes an example system that includes "a general purpose computing system environment, such as compute node 100." *Id*., 4:27-29.

174. "[C]ompute node 100 typically includes at least one processing unit 102 and memory 104" (*e.g.*, RAM, ROM, and/or flash memory). *Id*., 4:29-36.

175. The system may also have additional common features/functionality, such additional removable 108 and/or non-removable 110 storage (*e.g.*, disks, tapes), communications connections 112, input devices (*e.g.*, keyboard) 114, and output devices (*e.g.*, monitor) 116. *Id*., 4:36-5:10. *See also* §VII.A above.

176. "Memory 104, removable storage 108, and non-removable storage 110 are all examples of computer storage media" which "includes, but is not limited to,

RAM, ROM, flash memory, DVDs, …, magnetic storage devices, or "any other medium which can be used to store the desired information and which can be accessed by compute node 100." EX1001, 4:46-55, Fig. 1 (reproduced below).

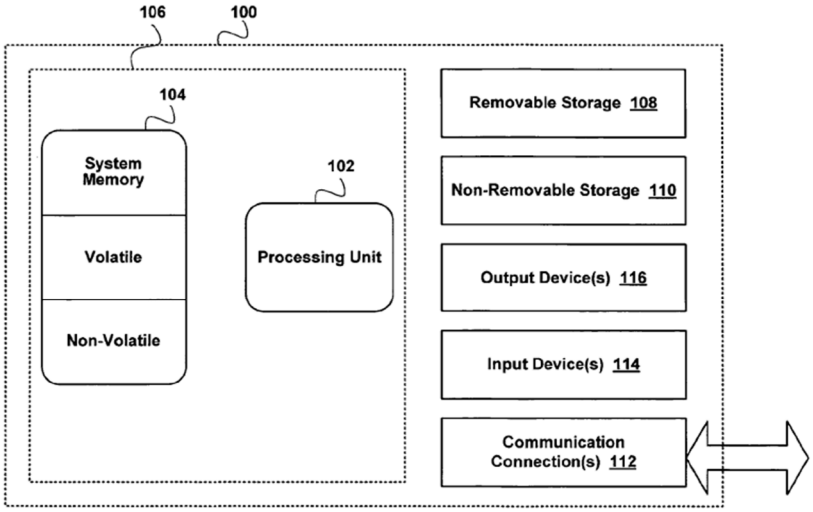


FIG. 1

'844 Patent, Figure 1

177. The '844 Patent then describes components of the system. "FIG. 2 is a diagram of a system 200 for root image caching and indexing in a block-level distributed application environment, in accordance with various embodiments of the present disclosure. In one embodiment, system 200 is implemented in a multi-computer system, such as an HPC cluster. In one embodiment, the application environment includes an operating system. In other embodiments, the application environment may contain other applications." EX1001, 5:12-26, Fig. 2 (reproduced below).

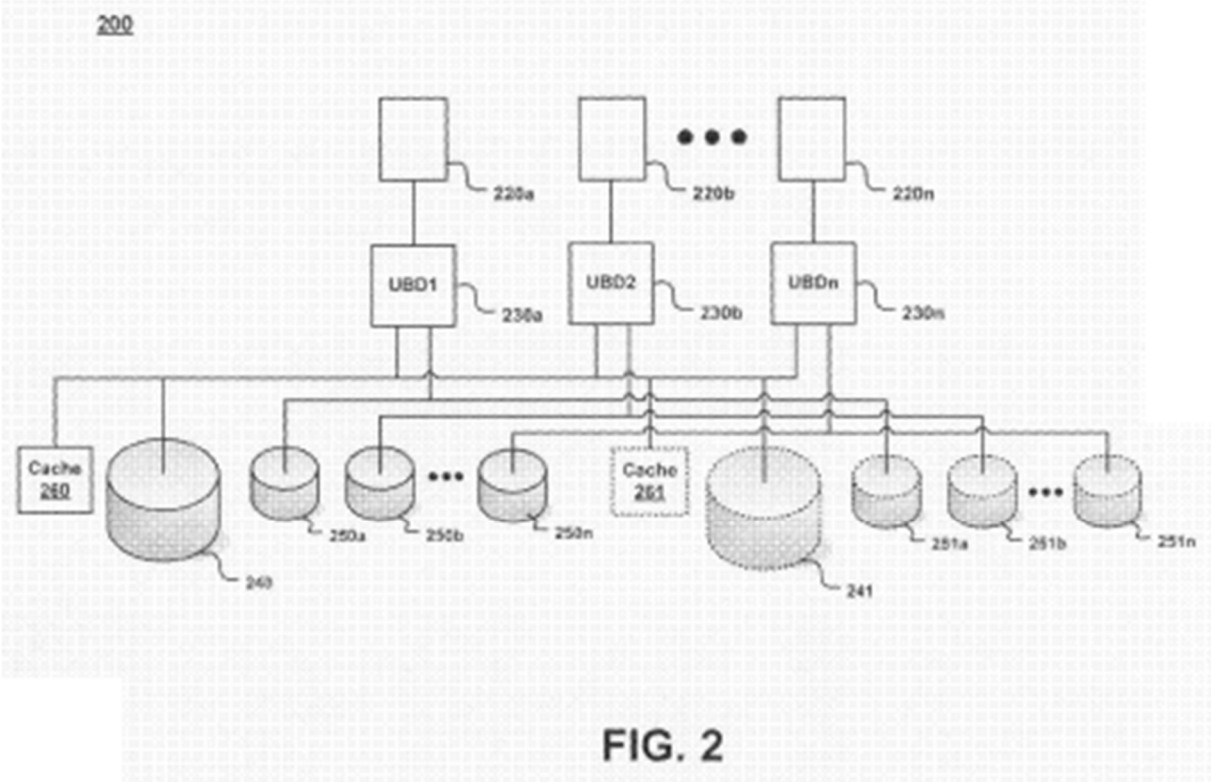


FIG. 2

'844 Patent, Figure 2

178. It then describes how the nodes and the storage they use. "System 200 has a number of compute nodes 220*a-n* coupled to first storage unit 240 and a corresponding second storage unit 250*a-n* [sic] though a corresponding union block device (UBD) 230*a-n*. To the compute nodes 220*a-n*, it appears that they have access to their own version of a distributed application environment. However, a separate and complete boot image is not created and stored for each compute node 220*a-n*." *Id*. Of note is the disclosed "UBD", which I describe in more detail in §VIII.C below.

179. Instead of having separate boot images for each node, "a first storage unit 240" "store[s] blocks of a root image of an application environment. The root

image contains data initially common to the compute nodes 220*a-n*" and "is not changed by the compute nodes 220*a-n*." *Id*., 5:27-32.

180. In one embodiment, compute nodes 220*a-n* have read-only access to the first storage unit 240. Each compute node 220*a-n* also has a corresponding second storage unit 250*a-n* for storing a leaf image, which in the challenged claims includes "only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of [said/the] root image" or similar language. *Id*., 5:33-34, 10:61-65, 11:32-35, 12:4-8, 12:47-50.

181. In one embodiment, "each compute node 220*a-n* has a corresponding second storage unit 250*a-n* for storing a leaf image. The first storage unit 240 and second storage units 250*a-n* may each be contained on separate physical storage devices, on separate logical spaces on the same storage device, or any combination thereof." EX1001, 5:33-58.

182. "The leaf image may contain blocks of new data, blocks of changed data, or other blocks of data unique to the individual compute node." *Id*. Thus, "a leaf image will describe the changes made by the respective compute node 220*a-n* to its instance of the application environment." *Id*.

183. One main reason why the challenged claims were allowed is the addition of "only" in "only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of [said/the]

root image" or similar language. *Id*., 5:33-34, 10:61-65, 11:32-35, 12:4-8, 12:47-50.

184. The '844 Patent states that "[w]ith respect to changes to the root image, only the specific blocks that are changed are stored in the leaf image. … One compute node (e.g., compute node 220*a*) desires to make a change to this file which involves a modification of only a few specific blocks of the file (e.g., blocks 4-9). In this example, only the modified blocks (e.g., blocks 4-9) will be stored in the compute node's leaf image (e.g., leaf image stored on second storage device 250*a*) plus some small overhead." EX1001, 5:33-58.

185. This claim language differs from the specification, which states that the leaf image "includes, but is not limited to, new data blocks for the compute node and blocks of the root image that the compute node has changed." *Id*., 7:64-66.

186. The first storage unit 240 and second storage units 250*a-n* may each be contained on separate physical storage devices, on separate logical spaces on the same storage device, or any combination thereof. *Id*., 5:34-38.

187. I note that Claim 23 requires that the "leaf image portion" comprise "only additional data blocks not previously contained in said root image portion and changes made by said first compute node to the blocks of said root image." *Id*., 13:8-12. A POSITA would have understood this to claim snapshots as I describe in §VII.E above.

188. Next, the ’844 Patent describes a “union block device” (UBD) that logically merges blocks between a root image and one or more lead images, just as snapshotting systems do. I describe this UBD and its relationship to commonly known snapshots in detail in §VIII.C below.

189. The ’844 Patent describes an embodiment that uses caching to improve performance. “[T]he present disclosure provides for caching of portions of the root image. Because several compute nodes in a cluster may often access the same data (e.g., same drivers, same library files, etc.) on the root image, tremendous speed improvements can be realized by caching such data in cache 260.” EX1001, 6:38-49. A POSITA would have known that caching can be added to many systems to improve performance and has been in operation for many years, as I detail in §VII.D above.

190. Moreover, the ’844 Patent describes an embodiment where “the cache may be contained within a single storage appliance, along with the first storage unit 240 and the second storage units 250*a-n*.” EX1001, 6:38-49. A POSITA would have known that sharing any data (volatile or non-volatile caches or otherwise) was also well known in the art. *See* §VII.C above.

191. Finally, the ’844 Patent describes an embodiment “for indexing the root image” because “it is beneficial for each compute node to have access to an index of its file system.” EX1001, 7:18-25. A POSITA would have known that indexing of

any data was also well known in the art, as I describe in §VII.F above. The '844 Patent further describes how indexing can be shared with multiple nodes: again, a POSITA would have known how to share data across the network. *See* §VII.C above. I discuss the '844 Patent's indexing disclosures in more detail in §VIII.B below.

B. The '844 Patent Discloses Indexing at the Block Layer

192. The '844 Patent claims "indexing" but does not provide sufficient details as to *how* the indexing is performed. A POSITA would have known that there are multiple forms in which indexes can be organized. *See* §VII.F above.

193. The '844 Patent further discloses that indexes can be shared with others. "It is appreciated that the indexing results can be provided to the other compute nodes in a number of ways. For example, the results may be stored on a shared storage unit (such as, but not limited to, cache 260), the results may [sic] provided directly to the other compute nodes, etc." EX1001, 7:37-41. I note that the '844 Patent does not say how such index data can be shared with others. A POSITA, however, would have known that data can be easily shared across the network using block-layer (*e.g.*, iSCSI) or file-layer protocols (*e.g.*, NFS). *See* §VII.B.1 above.

194. For any program to be able to index files at the file system layer, it would require access to a fully mountable, complete file system. As described above (*see* §VII.B above), this requires access to a base image plus one or more snapshots

that provide a complete view of the file system at a point in time. A POSITA would have also understood that merely taking some subset of blocks (*e.g.*, an incremental snapshot) does not provide a valid format file system; an incremental "leaf" snapshot alone cannot even be mounted or accessible as a file system because critical data structures that make up files, folders, and names are missing.

195. The '844 Patent describes that indexing takes place at the logical block layer, not at the file system layer, for four reasons.

196. First, the '844 Patent states that file systems are "agnostic" to the fact that their lower-level blocks and sectors are spread around a root image plus one or more leaf images (*i.e.*, snapshots). "Step 520 involves the compute node indexing the root image portion of its file system. In such a case, the compute node is to some extent agnostic to the fact that its file system is divided between a root image portion and a leaf image portion." EX1001, 9:48-52. A POSITA would have understood this passage to mean that indexing takes place at either the root image or one of the leaf images.

197. Second, the '844 Patent clearly states that it indexes only the leaf image (an incremental snapshot) independently from file systems. "Further down the road, the benefits of indexing the root image separately from the leaf image are realized when the compute node requires that its file system be re-indexed. At step 560, the compute node re-indexes its file system by re-indexing its corresponding leaf image

portion with the previous indexing results of the root portion. Thus, by operating at the block level, embodiments of the present disclosure provide file system and operating system independent systems and methods for distributing an application environment to a compute node." EX1001, 9:62-10:4. A POSITA would have understood that this can only take place by indexing at the block layer. Indeed, it would be impossible to index files in a leaf image (incremental snapshot) because it will not even mount as a file system volume.

198. Third, the '844 Patent reinforces the purported benefit of being able to quickly index only the leaf image because it is small. "Thereafter, the compute nodes 220*a-n* need only be concerned with indexing their respective leaf image, which are relatively small compared to the root image." EX1001, 7:31-33. This further reinforces that the idea that the '844 Patent's intended indexing is of the data blocks/sectors of the root/leaf images, which can only take place at the block layer.

199. Fourth, the '844 Patent's claim 19 claims in its preamble "a method for indexing file systems" but its actual limitation not once mentions a file, file name, or file system structure. Rather, it mentions "data blocks" of root and/or leaf images. A POSITA would have understood that the indexing here operates at the block/image level; that is, when the preamble states "indexing file system," a POSITA would have understood it to be nothing more than an outcome borne out of the fact that file systems are logically structured above block devices. Stated

differently, every file that a user or application can access (*e.g.*, by name), eventually translates to one or more disk blocks at the block layer; thus, it is natural to discuss (indexed) blocks that relate to files and hence their file systems, even when actual indexing operations take place at the block layer. *See* §VII.B above.

C. The '844 Patent's Union Block Device (UBD)

200. I note that the '844 Patent does not define what a UBD is. The patent describes that UBDs may be low-level drivers. "In one embodiment, UBDs 230*a-n* are effectively low-level drivers that operate as an interface between the first and second storage devices and the file system of each compute node 220*a-n*. … Because UBDs 230*a-n* operate below the file system, they are concerned merely with the blocks of data themselves, rather than files they form." EX1001, 5:60-6:1. Many storage drivers reside below file systems. *See* §VII.B above.

201. UBDs are said to "determine what leaf image (from the appropriate second storage unit 250) is needed for portions of the application environment that their respective compute nodes 220*a-n* have changed. UBDs 230*a-n* also locate the portions of the application environment that are not changed by their respective compute nodes 220*a-n*. These portions may reside in the root image. There may also be intermediate images (not depicted in FIG. 2) comprising versions of the root image from which a compute node's instance of the application environment is

derived." EX1001, 6:4-8. A POSITA would have understood that this is no more than common traversal of snapshots. *See* §VII.E above.

202. Moreover, "UBDs 230*a-n* may also modify the leaf image in response to their respective compute node's access to its instance of the application environment. For example, upon receiving a write request from their respective compute nodes for a sector X, the UBDs 230*a-n* will create an appropriate persistent mapping for sector X and then write sector X onto their respective second storage units 250*a-n*, where sector X can then be modified." EX1001, 6:20-27. Again, a POSITA would have understood that the UBD functionality disclosed here is nothing more than typical copy-on-write (CoW). *See* §VII.E above.

203. Lastly, UBDs logically merge different root and leaf images, which are nothing more than snapshots. It is also natural to extend them to integrate data stored in a cache, which is yet another sort of snapshot in that it contains a subset of the data from the backing store (see §VII.D above). Thus "Step 330 involves merging the blocks of the root image, the blocks of the leaf image, and the relevant blocks of the cache, if any, to create the application environment. In other words, the merging occurs at an operational level between the file system of a compute node and the first storage unit, the corresponding second storage unit, and the cache. Once the application environment has been created, it will appear to the compute node as one cohesive image rather than a base image plus its additions, deletions, and

modifications. To the compute node, it appears that it has access to its own unique version of an application environment. However, a separate and complete boot image is not actually stored for the compute node." EX1001, 8:13-25. A POSITA would have recognized this merging functionality to be no different than what prior art snapshot systems already disclosed. *See* §VII.E above.

D. Prosecution History

1. The '477 Application

204. U.S. Application No. 11/709,477 (the "'477 Application"), which matured into the '844 Patent, was a continuation-in-part of U.S. Patent Application No. 11/395,816 (the "'816 Application"), which was a continuation-in-part of U.S. Patent Application No. 11/026,622 (the "'622 Application").

205. The '477 Application contained five independent claims (claims 1, 7, 14, 19, and 23). The first four independent claims stated that the leaf images comprised "additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of [said/the] root image." The fifth independent claim (claim 23) stated that the leaf images comprised "additional data blocks not previously contained in said root image portion and changes made by said first compute node to the blocks of said root image." EX1002 (the '844 Patent File History), 25-30.

206. The PTO issued a First Office Action on January 5, 2010. The Examiner rejected claims 1-13 as obvious over U.S. Patent Application Publication No. 2003/0126242 A1 ("Chang") in view of U.S. Patent No. 6,101,576 ("Kobayashi"). EX1002 (the '844 Patent File History), 59-64. Kobayashi was only cited for teaching physical caches, and the Examiner found it would have been obvious to a POSITA at the time of the alleged invention to have combined the "detailed image/data cache" teachings of Kobayashi with the "server/client image [sic] assess and remotely stored boot/OS system" teachings of Chang "for the benefit of increasing the speed of data transfers from the host to client in data streaming or transfer operations as frequently accessed data is kept in high speed memory and not constantly retrieved from slower disk sources." *Id*., Claims 14-27 (*i.e.*, all remaining claims) were rejected as anticipated by Chang. *Id*., at 53-58.

207. The Applicants submitted an Amendment and Response on April 5, 2010. Therein, the first four independent claims (claims 1, 7, 14, and 19) were amended to include a new "wherein" limitation (immediately following the "comprising" clause discussed above), stating that the "leaf images [of/for] respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes" or similar language. For example, claim 7 was amended as follows:

7. (Currently amended) A method for providing data to a plurality of compute nodes, comprising:

storing blocks of a root image of said compute nodes on a first storage unit;

storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.

208. Claim 23 was also amended to include a new "wherein" limitation (also following the above-discussed "comprising" clause), stating that the "leaf image portion does not include blocks of said root image that are unchanged by said first compute node." EX1002 (the '844 Patent File History), 73-78.

209. In the accompanying remarks, the Applicants stated the addition of these new "wherein" clauses overcame the Examiner's January 5, 2010, anticipation and obviousness rejections given one or more purported gaps in the teachings of Chang. *Id*., at 82-88. The Applicants never contested the Examiner's findings with respect to the cache teachings of Kobayashi or the stated motivation to combine Kobayashi with Chang. Nor did the Applicants ever expressly raise any of the dependent claim limitations as a grounds for allowance. *Id*.

210. The PTO issued a second Office Action on May 25, 2010. The Examiner again rejected all 27 pending claims, largely citing the same anticipation

and obviousness grounds (but adding citations to Chang addressing the amended "wherein" language). EX1002 (the '844 Patent File History), 102-25.

211. The Applicants submitted an Amendment and Response on August 31, 2010. Therein, the first four independent claims (claims 1, 7, 14, and 19) were again amended, this time to include the term "only" immediately before the "comprising" clause discussed above, so as to now state that the claimed leaf images comprise "only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of [said/the] root image…" EX1002 (the '844 Patent File History), 164-83. For example, claim 7 was amended as follows:

> 7. (Currently amended) A method for providing data to a plurality of compute nodes, comprising:
>
> storing blocks of a root image of said compute nodes on a first storage unit;
>
> storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
>
> caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.

212. Claim 23 was also amended (for a second time) to now state that the claimed leaf images comprise "only additional data blocks not previously contained

in said root image portion and changes made by said first compute node to the blocks of said root image…" EX1002 (the '844 Patent File History), 166-71.

213. In the accompanying remarks, the Applicants stated these August 31, 2010, amendments overcame the Examiner's May 25, 2010, anticipation and obviousness rejections because the "client image copies" in Chang (what the Examiner had reportedly identified as the leaf images) include "blocks of the root image that are unchanged by a compute node (*e.g.*, the common OS and application blocks for storing the base boot image)." EX1002 (the '844 Patent File History), 176-82. Again, the Applicants never contested the Examiner's findings with respect to the cache teachings of Kobayashi or the stated motivation to combine Kobayashi with Chang. Nor did the Applicants explicitly raise any of the dependent claim limitations as a ground for allowance. *Id*.

214. Applicants also stated a UBD is "a driver implemented by and for a particular compute node;" that they do not need a separate processor; that "drivers serving as interfaces between operating system file systems for a computer system and associated storage devices are well known in the art;" and that a POSITA "would understand how a driver may serve as an interface between storage devices and a file system of a computer system." *Id*., 172-73.

215. The PTO issued a third Office Action on March 12, 2012. However, the Examiner never mentioned, let alone cited, Chang. Instead, the Examiner

rejected: (1) claims 1-13 as obvious over U.S. Patent No. 7,870,106 ("Nguyen") in view of Kobayashi; (2) claims 14-27 as being obvious over Nguyen; and (3) claims 19-27 as anticipated by Nguyen. EX1002 (the '844 Patent File History), 192-206. Again, Kobayashi was only cited for teaching physical caches, and the Examiner found it would have been obvious to a POSITA at the time of the alleged invention to have combined the "detailed image/data cache" teachings of Kobayashi with the "server/client image assess and remotely stored boot/OS system" teachings of Nguyen "for the benefit of increasing the speed of data transfers from the host to client in data streaming or transfer operations as frequently accessed data is kept in high speed memory and not constantly retrieved from slower disk sources." *Id*., The Office Action concluded by stating that the Applicants' prior arguments were considered, but "are moot because the arguments do not apply to any of the reference[s] being used in the current rejection." *Id*., at 206.

216. The Applicants subsequently filed an Amendment on July 11, 2012. Therein, the Applicants overcame the Examiner's March 12, 2012, obviousness and anticipation rejections by stating that Nguyen was not prior art under 35 U.S.C. §103(a) given its alleged common ownership with the pending application. EX1002 (the '844 Patent File History), 224, 300. The Applicants also amended several claims. This included the substitution of "including" for "comprising" in independent claims 1 and 7. *Id*., at 239-45. Once again, the Applicants never

contested the Examiner's findings with respect to the cache teachings of Kobayashi or the stated motivation to combine Kobayashi with Nguyen. Nor did they raise any of the dependent claim limitations as a reason for allowance. *Id*., at 221-25.

217. The Applicants later stated, in their November 6, 2012, Comments on Statement of Reason for Allowance, that "[t]he amendments submitted on July 11, 2012, were directed to matters of form and to improve the clarity of the claims, and were not required to overcome the art of record." *Id*., at 305.

218. Thus, the sole purported point of novelty for the claims of the '844 Patent expressly raised during prosecution is the limitation, in each of the independent claims, that the leaf images can only comprise or include "additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of [said/the] root image" (or the language in claim 23 added by the Applicants at the same time). A POSITA would have understood this additional language to claim snapshots as I describe in §VII.E above.

2. The '622 Application

219. A different Examiner was assigned to the "grandparent" '622 Application. That Examiner rejected all of the claims initially proffered by the Applicants (who differed, in part, from the Applicants in the '477 Application) in an April 13, 2009, first Office Action—as being anticipated by Chang. The Examiner thereafter repeatedly rejected all subsequently pending claims (in Office Actions

dated August 24, 2009; March 15, 2010; August 13, 2010; and December 6, 2010) as obvious over Chang in view of U.S. Patent No. 6,751,658 ("Haun"). EX1009 (the '622 Application File History), 50-63, 92-109, 144-167, 203-25, 259-85. Indeed, the Applicants ultimately abandoned the '622 Application with no claims ever being deemed allowable. *Id*., at 294-95.

220. The Examiner assigned to the '622 Application repeatedly rejected every pending claim despite the prosecuting attorney (who was the same attorney prosecuting the '477 Application, at least for a significant portion of the time) making several amendments and arguments that resembled those offered in connection with the '477 Application (including the addition of the word "only" to several claims in a June 15, 2010 Amendment and Response). EX1009 (the '622 Application File History), 183-99. For example, independent claim 1 was amended to recite a "plurality of second file collections <u>separate from the root file system, each</u> comprising <u>only</u> modified portions of said distributed applications for respective ones of said plurality of compute nodes…" *Id*., at 184.

221. Further, while the '622 and '477 Applications were co-pending for an appreciable amount of time, the Examiner assigned to the '477 Application (unlike the Examiner assigned to the '622 Application) never relied on Haun. *See* EX1002 (the '844 Patent File History), 66. In fact, the Examiner assigned to the '477 Application only mentioned Haun once (in the initial, January 5, 2010, Office

Action), and then only to state that Haun (and another reference) were prior art of record and were "not relied upon but is considered pertinent to applicant's disclosure." *Id*., at 64.

3. Priority Date

222. The '844 Patent claims priority to the '816 Application (filed March 30, 2006), which in turn claims priority to the '622 Application (filed December 30, 2004). EX1002 (the '844 Patent File History), EX1009 (the '622 Application File History), EX1010 (the '816 Application File History). I understand that the Petitioners assert the '844 Patent is not entitled to either priority claim, as neither the '816 Application nor the original '622 Application ever mention caching or indexing—which are expressly recited in each challenged claim. EX1001 (the '844 Patent), 10:55-14:17. In fact, the Applicants admitted during prosecution of the '844 patent that "indexing and indexing results are not discussed anywhere within Nguyen" (a reference that issued from a continuation-in-part of the '622 Application). EX1002 (the '844 Patent File History), 223. As stated above, my opinions have been guided by my appreciation of how a POSITA would have understood the claims and the specification of the '844 Patent at the time of the alleged invention, which I have been asked to initially consider as no earlier than December 30, 2004 (the filing date of U.S. Patent Application 11/026,622, EX1009).

IX. THE CLAIMS

223. The tables below include identifiers for the ’844 Patent claims and limitations, which are referenced throughout this Declaration.

Reference	Claim Limitation
Claim 1	
[1.0]	1. A system for providing data to a plurality of compute nodes, the system comprising:
[1.1]	a first storage unit configured to store blocks of a root image of said compute nodes;
[1.2]	a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
[1.3]	a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes.

Claim 2	
[2]	2. The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value.

Claim 3	
[3]	3. The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance.

Claim 4	
[4]	4. The system as recited in claim 1 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.

Claim 5	
[5]	5. The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.

Claim 6	
[6]	6. The system as recited in claim 1 wherein said first storage unit is read-only.

Claim 7	
[7.0]	7. A method for providing data to a plurality of compute nodes, comprising:
[7.1]	storing blocks of a root image of said compute nodes on a first storage unit;
[7.2]	storing leaf images for respective compute nodes on respective second storage units, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes

Claim 7	
	do not include blocks of said root image that are unchanged by respective compute nodes; and
[7.3]	caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.

Claim 8	
[8.1]	8. The method as recited in claim 7 further comprising: receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and
[8.2]	providing said first portion of said data to said at least one of said compute nodes from said cache memory.

Claim 9	
[9]	9. The method as recited in claim 8 further comprising: updating said cache memory based on said read request.

Claim 10	
[10.0]	10. The method as recited in claim 9 wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises:
[10.1]	caching said second portion in said cache memory; and
[10.2]	removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value.

Claim 11	
[11]	11. The method as recited in claim 7 further comprising:

Claim 11	
	merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments.

Claim 12	
[12]	12. The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory.

Claim 13	
[13]	13. The method as recited in claim 7 wherein said first storage unit is read-only.

Claim 14	
[14.0]	14. A system for indexing file systems for a plurality of compute nodes, the system comprising:
[14.1]	a first storage unit configured to store blocks of a root image of said compute nodes;
[14.2]	a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
[14.3]	a plurality of union block devices corresponding to said compute nodes, said union block devices configured to interface between said compute nodes and said first and second storage units to distribute said file systems to said compute nodes, wherein said union block devices are configured to create said file systems by

Claim 14	
	merging the blocks of said root image stored on the first storage unit with the blocks of respective leaf images stored on respective second storage units, and
[14.4]	wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.

Claim 15	
[15]	15. The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance.

Claim 16	
[16]	16. The system as recited in claim 14 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit and respective second storage units, said union block devices configured to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.

Claim 17	
[17]	17. The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.

Claim 18	
[18]	18. The system as recited in claim 14 wherein said first storage unit is read-only.

Claim 19	
[19.0]	19. A method for indexing file systems for a plurality of compute nodes, comprising:
[19.1]	storing blocks of a root image of said compute nodes on a first storage unit;
[19.2]	storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images for respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes;
[19.3]	merging the blocks of said root image with the blocks of respective leaf images stored on respective second storage units to create respective file systems for respective compute nodes;
[19.4]	receiving indexing results pertaining to said root image from one of said compute nodes; and providing said indexing results to the others of said compute nodes.

Claim 20	
[20]	20. The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit.

Claim 21	
[21]	21. The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the

Claim 21	
	compute nodes and said first storage unit and respective second storage units.

Claim 22	
[22]	22. The method as recited in claim 19 wherein said first storage unit is read-only.

Claim 23	
[23.0]	23. A computer-readable storage medium having instructions stored thereon that, in response to execution by at least one computing device, cause the at least one computing device to:
[23.1]	receive data blocks of a file system, said data blocks comprising a root image portion
[23.2]	and leaf image portion, said leaf image portion comprising only additional data blocks not previously contained in said root image portion and changes made by said first compute node to the blocks of said root image, wherein said leaf image portion does not include blocks of said root image that are unchanged by said first compute node,
[23.3]	wherein said file system is the result of merging said root image portion and said leaf image portion together at the block-level;
[23.4]	index said root image portion; and provide the results of said indexing to a second compute node, wherein said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node.

Claim 24	
[24]	24. The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to

Claim 24	
	store said results of said indexing on a shared storage unit accessible by said second compute node.

Claim 25	
[25]	25. The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion.

Claim 26	
[26]	26. The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.

Claim 27	
[27]	27. The computer-readable storage medium of claim 23 wherein said root image portion is read-only.

X. OVERVIEW OF THE PRIMARY PRIOR ART REFERENCES

224. I first note that the prior art references analyzed in the grounds below are analogous to the ’844 Patent because they are within the same field of endeavor and/or reasonably pertinent to one or more problems addressed by the ’844 Patent. For example, all are related to storage systems, snapshot management, and using copy-on-write techniques.

A. Sapuntzakis

225. Sapuntzakis describes a system that "shows how to quickly move the state of a running computer across a network, including the state in its disks, memory, CPU registers, and I/O devices" which it "call[s] this state a *capsule*." EX1005 (Sapuntzakis), Abstract.

226. These "developed techniques [] reduce the amount of data sent over the network" for example by using "copy-on-write disks [to] track just the updates to capsule disks." *Id*.

227. In Sapuntzakis, "[c]omputers and storage in the Collective system act as caches of capsules." EX1005 (Sapuntzakis), p. 2.

228. "As users travel, the Collective can move their capsules to computers close to them, giving users a consistent environment." *Id*.

229. "Capsules in the Collective system are seldom created from scratch, but are mostly derived from other capsules." *Id*., p. 5.

230. Sapuntzakis "store[s] the disks in these capsules efficiently by creating a hierarchy, where each child capsule could be viewed as inheriting from the parent capsule with the differences in disk state between parent and child captured in a separate copy-on-write (COW) virtual disk." *Id*., p. 5.

231. "At the root of the hierarchy is a *root disk*, which is a complete capsule disk. All other nodes represent a COW disk. Each path of COW disks originating

from the root in the capsule hierarchy represents a capsule disk; the COW disk at the end of the path for a capsule disk is its latest disk." *Id*., p. 5. A POSITA would recognize that Sapuntzakis's disclosures here mirror those of snapshotting systems. *See* §VII.E above.

232. Sapuntzakis's approach is similar to the idea of disk imaging, but it allows multiple capsules to exist (and be cached) on the same machine. "This also translates to efficient usage of the buffer cache of the host OS, when multiple capsules sharing COW disks simultaneously execute on the same host." *Id*., p. 5.

233. "Figure 1 shows an example of a capsule hierarchy illustrating how it may be used in a university. The root capsule contains all the software available to all students. The various departments in the university may choose to extend the basic capsules with department-specific software. The department administrator can update the department capsule by deriving new child capsules." *Id*., p. 5, Figure 1 (reproduced below).

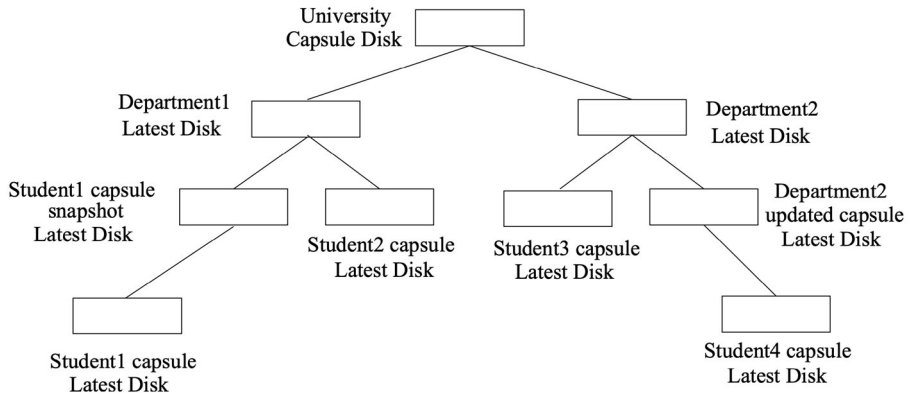


Figure 1: An example capsule hierarchy.

Sapuntzakis, Figure 1

234. Sapuntzakis describes how CoW disks/images are implemented efficiently. "Each COW disk is implemented as a bitmap file and a sequence of extent files. An extent file is a sequence of blocks of the COW disk, and is at most 2 GB in size (since some file systems such as NFS cannot support larger files). The bitmap file contains one bit for each 16 KB block on the disk, indicating whether the block is present in the COW disk. We use sparse file support of Linux file systems to efficiently store large yet only partially filled disks." *Id*., p. 5.

235. Sapuntzakis describes how the CoW disks/snapshots are transparent to applications (here, the VMM). "Figure 2 shows an example capsule disk and the chain of COW disks that comprise it. Note that the COW disk hierarchy is not visible to the VMM, or to the OS and applications inside the capsule; all of them see a normal flat disk as illustrated in the figure." *Id*., p. 6, Figure 2 (reproduced below).

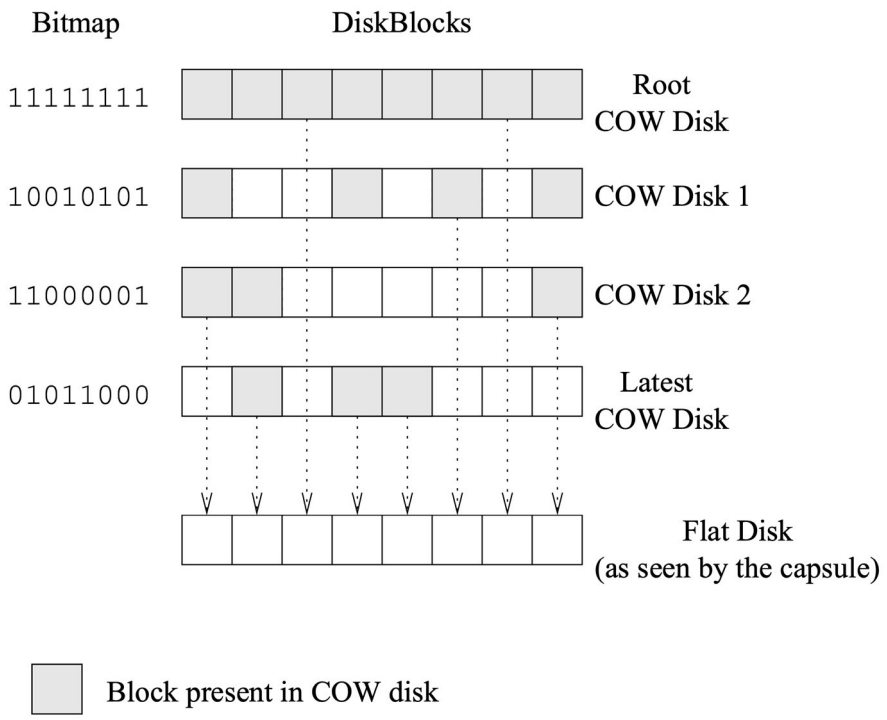


Figure 2: An example capsule disk.

Sapuntzakis, Figure 2

236. Sapuntzakis describes how CoW disks are stored and managed efficiently, including the deletion and merging (or "collapsing") snapshots together. "To avoid accumulation of costs in storing the intermediate COW disks, and the cost of looking up bitmaps, we have implemented a promote primitive for shortening these chains. We promote a COW disk up one level of the hierarchy by adding to the disk all of its parent's blocks not present in its own. We can delete a capsule by first promoting all its children and then removing its latest disk. We can also apply the promotion operations in succession to convert a COW disk at the bottom of the hierarchy into a root disk." *Id*., p. 6.

237. Sapuntzakis explains how the capsules are read and accessed efficiently, as needed, to speed up capsule start-up times. "To reduce the start-up time of a capsule, the COW disks corresponding to a capsule disk are read page-by-page on demand, rather than being pre-fetched. Demand paging is useful because COW disks, especially root disks, could be up to several gigabytes in size and prefetching these large disks could cause an unacceptable start-up delay. Also, during a typical user session, the working set of disk blocks needed is a small fraction of the total blocks on the disk, which makes pre-fetching the whole disk unnecessary. Most OSes have been designed to hide disk latency and hence can tolerate the latency incurred during demand paging the capsule disk." *Id*., p. 6. *See also* §VII.A above.

238. Sapuntzakis illustrates an implemented system using the aforementioned techniques. "The implementation of the capsule disk system, including demand paging, is shown in Figure 3. The shim library intercepts all of VMware's accesses to plain disks and forwards them to a disk server on the local machine. The disk server performs a translation from a plain disk access to the corresponding access on the COW disks of the capsule. Each COW disk can either be local or remote. Each remote COW disk has a corresponding local shadow COW disk which contains all the locally cached blocks of the remote COW disk." *Id*., p. 6, Figure 3 (reproduced below).

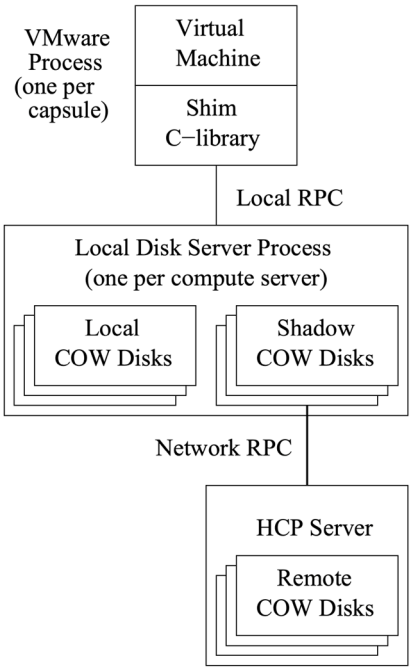


Figure 3: Implementation of capsule disks and demand paging.

Sapuntzakis, Figure 3

239. As I discussed in §VII.E above, snapshot-based systems use CoW and merge/unify snapshots as applications traverse one CoW snapshot to the next.

240. Sapuntzakis Figure 2 (reproduced above) shows this hierarchy of incremental/leaf images/snapshots backed by a "flat disk" (root/base image). "Figure 2 shows an example capsule disk and the chain of COW disks that comprise it." EX1005 (Sapuntzakis), p. 6.

241. Sapuntzakis discloses a "shim" software library that effectively merges the snapshots and base image. "The COW disk implementation interfaces with GSX Server through a shim library that sits between GSX Server and the C library. The shim library intercepts GSX Server's I/O requests to disk image files in VMware's

'plain disk' format, and redirects them to a local disk server. The plain disk format consists of the raw disk data laid out in a sequence of extent files. The local disk server translates these requests to COW disk requests, and executes the I/O operations against the COW disks." EX1005 (Sapuntzakis), p. 6. I note that, like the UBD as described in the '844 Patent, Sapuntzakis's shim also integrates blocks from the cache (the Shadow COW disks) into its cohesive images formed primarily from the Root COW disks and the Local COW disks. *Id*.

242. A POSITA would have known that in Sapuntzakis, the GSX Server was the application, and the disk image files resided on a local storage of a Linux operating systems. "Our prototype's platform uses VMware GSX Server 2.0.1 running on Red Hat Linux 7.3 (kernel 2.4.18-10) to execute x86 capsules." EX1005 (Sapuntzakis), p. 2. Therefore, Sapuntzakis's shim library acts as a driver that merges I/O requests from an application to the native Linux storage devices.

243. In sum, Sapuntzakis discloses data structures and methods that a POSITA would have found functionally identical to the '844's UBD, especially in light of the '844 patent's description of what a UBD may be and how it reportedly works (*see* §VIII.C above). Moreover, Sapuntzakis operates and integrates with caches as discussed above. Lastly, a POSITA would have known that any computer systems, storage system, and more integrate with caches: that is, the software

systems (*e.g.*, operating systems, applications) often consult the fast caches first, then the underlying (and typically slower layers), as I detail in §VII.D above.

B. Holzmann

244. Holzmann's "invention is directed to a system and method for booting multiple servers or other network resources from a single operating system image." EX1007 (Holzmann), Abstract. "The operating system image is stored on a solid state disk" to improve performance. *Id*.

245. Holzmann utilizes common caching. "When a server is booted, cache space is allocated in the volatile memory portion of the solid state disk. This cache is used to store data necessary for booting and operation of the operating system. As additional servers or other network resources are booted, the cache is used to access the necessary operating system data." *Id*.

246. Holzmann's "FIG. 1 is a block diagram illustrating the component parts of the invented system and the functions of each component part" which "comprises a solid state disk system 101 having a storage means 102, a control module 103, a memory module 104, an interface module 105 that communicates with external devices 106, and an internal power source 107." *Id*., ¶[0017], FIG. 1 (reproduced below).

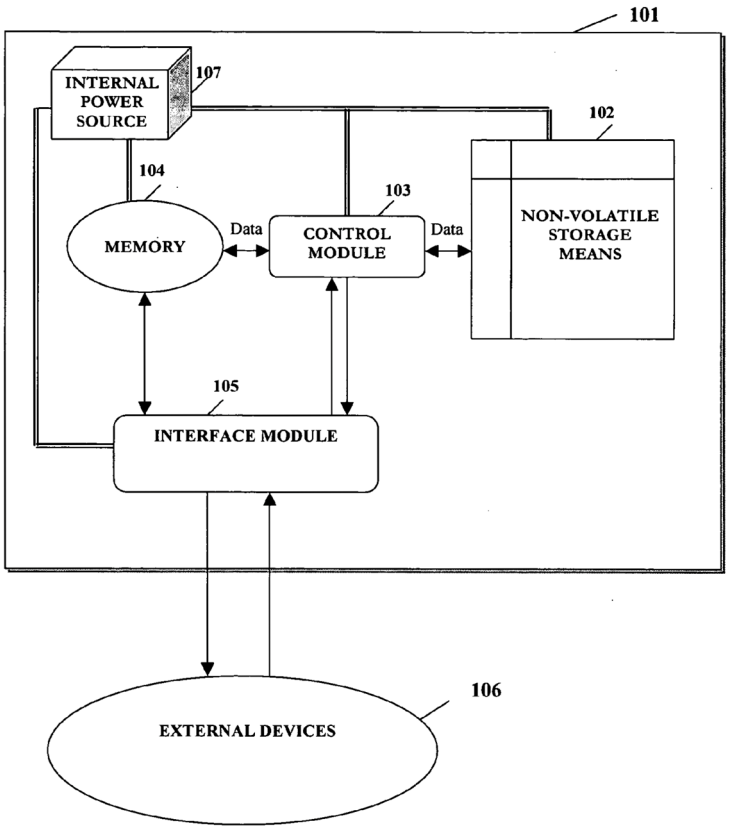


Fig. 1

Holzmann, Figure 1

247. Holzmann's storage can include different types of devices. "The storage means 102 may comprise, for example, a hard disk system. The storage means may alternatively comprise another non-volatile storage means, such as a semiconductor memory array or flash memory array." *Id.*, ¶[0018].

248. Holzmann's "control module 103 facilitates the copying of data to the storage means 102 from the memory module 104, and the writing of data from the storage means 102 to the memory module 104. In the present invention, the control

module 103 automatically manipulates the copying and writing of data between the memory module 104 and the storage means 102." *Id*., ¶[0019]

249. Holzmann's "memory module 104 is more quickly accessible, and performs read and write processing functions more quickly, than non-volatile or disk memory, such as storage means 102" and "preferably comprises at least one random-access memory (RAM) module", "dynamic random-access memory (DRAM), synchronous DRAM (SDRAM), double data rate (DDR) memory, or other appropriate memory technology." *Id.*, ¶[0020].

250. Holzmann shows how in "FIG. 4, the memory module 104 may contain an image of an operating system." *Id*., ¶[0020], FIG. 4 (reproduced below).

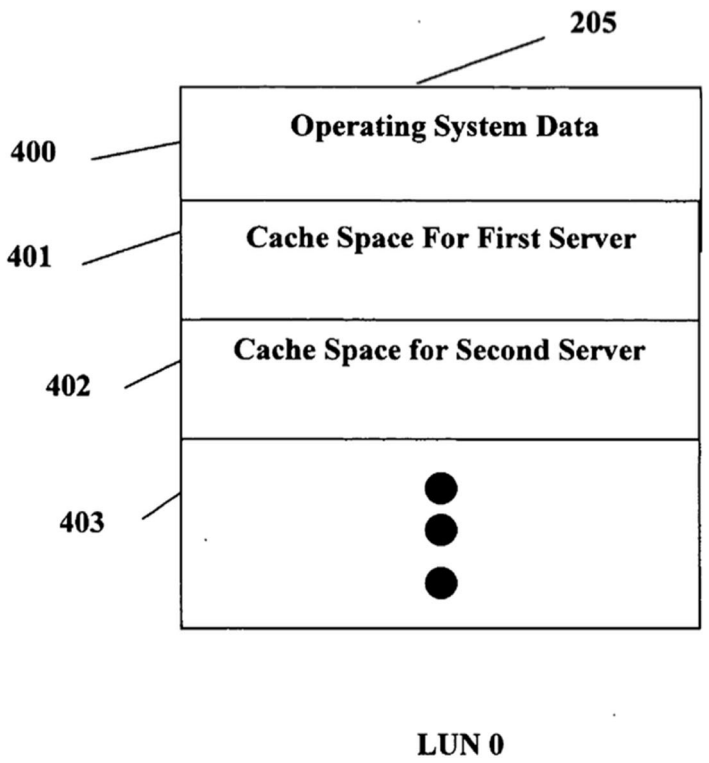


Fig. 4

Holzmann, Figure 4

251. "Alternatively, the memory module 104 may contain only those parts of the operating system image required for server operation." *Id*.

252. Then, "the control module 103, monitors boot requests from external devices 106. Upon receiving a boot request, the control module 103 loads the required portion of the operating system image (the portion loaded typically depends on the operating system and its boot process) into memory module 104, if the image has not been loaded freely." *Id*.

253. Holzmann uses caching as is well-known in the art. "The control module 103 then allocates a sub-portion of the memory portion to be used as a cache for the operating system. As multiple servers or other external devices 106 are booted, the control module 103 allocates additional space into the cache stored in memory module 104. Portions of the cache may be assigned statically for each server or other external devices 106. Alternatively, and preferably, the portions of the cache attributed to each server may be assigned dynamically, as described further below with reference to FIG. 5." *Id*., ¶[0021]. *See also* §VII.D above.

254. Holzmann's "interface module 105 manages communication between the system 101 and external devices 106. The interface module 105 receives and processes read and write commands from external devices 106" which then "issues commands to the control module 103 and to the memory module 104, as the case may warrant. The interface module 105 then receives data and commands from the control module 103 and the memory module 104, and returns requested data or otherwise responds to the external devices 106." EX1007 (Holzman), ¶[0022].

255. Next, Holzmann describes specific actions upon receiving data read/write requests: "the interface module 105 receives data requests (read commands) from external devices 106 for particular data blocks. The interface module 105 translates these blocks into segments, retrieves the data segments from memory 104, and returns the requested data blocks to the external devices 106. The

interface module 105 also receives write commands from external devices 106 to update data disposed on the memory module 104. The interface module sends corresponding write data to the memory module 104 for updating the data." *Id.*, ¶[0023].

256. Finally, Holzmann states that "at least one of the external devices 106 requires booting from an operating system" which "requests operating system data through the interface module 105." *Id.*, ¶[0024].

257. That "control module 103 monitors the request for operating system data, and determines whether the particular operating system data block requested by the external devices 106 has previously been loaded into the operating system cache. If the data block has been loaded into the cache, and that data block has not been overwritten subsequent to being loaded into the cache, the requested data block is provided from the cache in memory module 104 to interface module 105 for transmission to the appropriate external device 106." *Id.*

C. Federwisch

258. Federwisch (EX1008) discloses a method for efficiently determining and transmitting changes in snapshots. Federwisch is assigned to Network Appliance ("NetApp"), who created storage filers that use copy-on-write snapshots, writable and mirrored snapshots, and exported using network protocols. *See* EX1008 (Federwisch), Abstract. *See also* §VII.E above.

259. Federwisch discloses a "system and method for remote asynchronous replication or mirroring of changes" between "a source file system snapshot" and "a destination replica file system." EX1008 (Federwisch), Abstract.

260. Federwisch compares two snapshots "using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file system, which identifies changed blocks in the respective snapshot files based upon differences in volume block numbers identified in a scan of the logical file block index of each snapshot." EX1008 (Federwisch), Abstract. Thus, every snapshot in a volume has a "logical file block index."

261. Federwisch, like any computer system, stores and caches data in fast memory. *See* §VII.D above. When a file has to be loaded for a user or application, if the data is not already cached in memory, then it is retrieved from the actual storage devices, by locating the index to that file's blocks.

262. "Bridging the disk software layers with the network and file system protocol layers is a file system layer 450 of the storage operating system 400. Generally, the layer 450 implements a file system having an on-disk format representation that is block-based using, e.g., 4-kilobyte (KB) data blocks and using inodes to describe the files. In response to transaction requests, the file system generates operations to load (retrieve) the requested data from volumes if it is not resident 'in-core', i.e., in the filer's memory 325. If the information is not in

memory, the file system layer 450 indexes into the inode file using the inode number to access an appropriate entry and retrieve a volume block number." EX1008 (Federwisch), ¶[0053].

263. Federwisch compares the indexes to file's block numbers between two snapshots to identify whether data has been changed between the snapshots (as would be accomplished using the CoW techniques discussed in §VII.E above).

264. "The existence of a changed volume block number at a given offset in the index between the base snapshot inode file and incremental snapshot inode file generally indicates that one or more of the underlying inodes and files to which the inodes point have been changed." EX1008 (Federwisch), ¶[0071].

265. Federwisch scans these indexes to identify unchanged blocks. "A scanner 820 searches the index for changed base/incremental inode file snapshot blocks, comparing volume block numbers or another identifier. … Hence, scanned blocks at the same index in both snapshots can be efficiently bypassed, reducing the scan time." EX1008 (Federwisch), ¶[0072].

266. A POSITA would have understood these disclosures to mean that in Federwisch, each snapshot (corresponding to the '844 Patent's "leaf image") has its own index of file blocks. "More specifically, each entry in the inode map 1400 contains an entry for each inode on the source snapshot. Each inode entry 1402 in

the map is indexed and accessed via the source inode number (1404)." EX1008 (Federwisch), ¶[0109].

267. When snapshots in Federwisch are copied from a source to a destination, although the data has not changed, the snapshot's block locations may have changed. *See* EX1008 (Federwisch), ¶¶[0015], [0061]. Thus, Federwisch updates the destination snapshot's index maps, a form of re-indexing.

268. "Fig 19 shows an exemplary new inode map file 1900 generated from the old inode map file 1800 of Fig. 18 in accordance with the flip procedure 1700. The new map includes fields for the new source (old destination) inode 1902, new destination (old source) inode 1904, new source (old destination) generation number 1906 and new destination (old source) generation number 1908. As a result of the flip, the entry 1910 for new source inode 150 is presented in appropriate index order and is paired with new destination inode 190 (and associated generation numbers)." EX1008 (Federwisch), ¶[0134].

269. Federwisch provides another example, where "the Destination Snapshot A (2002) is now prepared to transfer changes so as to establish a mirror in Destination Snapshot B (2004). … In this case, both the new source and the new destination system have their own inode maps A and B (2012 and 2014), indexed off the old source 2001, and referencing the inodes in their respective trees. … It then treats the new source as the appropriate map index for the new associated inode

map 2018. In the associated map, it stores the new source generation number for the new source index inode number, with each index entry also associated with/mapped to the new destination inode/generation number extracted from the inode map B (2014)." EX1008 (Federwisch), ¶[0137].

270. Finally, the NetApp filer that Federwisch discloses include a non-volatile NVRAM cache. "The NVRAM is filled, in parallel with the buffer cache, after each client request is completed, but before the result of the request is returned to the requesting client." EX1008 (Federwisch), ¶[0048]. *See also* §VII.E above.

271. In Federwisch, snapshot data (the "leaf image" in the '844 Patent) as well as active file systems ("root" image in the '844 Patent) are both cached on demand. "The snapshot is stored on-disk along with the active file system, and is called into the buffer cache of the filer memory as requested by the storage operating system 400 or snapshot mirror application 490 as described further below." EX1008 (Federwisch), ¶[0058].

272. Indeed, Federwisch employs typical caching as I describe in §VII.D above. "The inode files may each be loaded into the buffer cache of the source file server memory from the on-disk versions thereof using conventional disk access processes as directed by the storage operating system snapshot manager (480 in Fig. 4)." EX1008 (Federwisch), ¶[0070].

273. Federwisch recognizes the usefulness of sharing files and data in a format-agnostic manner to other systems. "More generally, the format should also be independent of the underlying network protocol or device (in the case of a tape or local disk/non-volatile storage) protocol and file system—that is, the information is system 'agnostic,' and not bound to a particular operating system software, thereby allowing source and destination systems of different vendors to share the information. The format should, thus, be completely self-describing requiring no information outside the data stream." EX1008 (Federwisch), ¶[0086].

274. Being a network appliance, Federwisch supports the ability to share files, snapshots, and whole volumes across the network (*e.g.*, using NFS or CIFS). *Id.*, ¶¶[0047], [0052], [0053]. [0057], [0126]. This sharing is called "an export." "The destination file system is exported read-only the user." *Id.*, ¶[0015]. This first snapshot may be an exported snapshot of the replicated snapshot on the destination. *Id.*, ¶[0120].

D. Birse

275. Birse describes a system "for supplying a reliable and maintainable operating system in a net-booted environment. According to one embodiment, a network computer (NC) system including an NC server and multiple NC clients is managed by an NC client causing the remainder of the NC clients that are subsequently booted to receive operating system software that is configured

differently than that currently in effect by replacing one or more system volumes on the NC server containing the operating system software with one or more different system volumes." EX1006 (Birse), Abstract.

276. In Birse, "[a]n 'image' refers to an electronic representation of an individual disk or a portion thereof." *Id*., 4:1-2.

277. In Birse, a "'thick client' generally refers to a computer system, such as a personal computer (PC), that includes a hard drive or other local storage medium. Traditionally, such computer systems boot and execute application programs from the local storage medium." *Id*. Conversely, a "'thin client' generally refers to a disk-less computer system that relies upon external means, such as a server, to supply it with an operating system and application programs." *Id*.

278. In Birse, "the terms 'network computer client' or 'NC client' generally refer to a client (thick or thin) that boots by accessing a copy of the operating system over a network. As such, an NC client is not required to be disk-less." *Id*.

279. Moreover, a "'network computer server' or 'NC server' generally refer to a computer system that services requests of NC clients. For example, an NC server may provide access to a copy of the operating system to an NC client in response to a boot request." *Id*.

280. Then, a "'network computer system' or 'NC system' refers to a network including one or more NC clients and one or more NC servers." *Id*.

281. Birse's "'volume' generally refers to a fixed amount of storage on a storage medium, such as a disk or tape. In some instances, however, the term may be used as a synonym for the storage medium itself. However, it is possible for a single storage medium to contain more than one volume or for a volume to span more than one storage medium." *Id*.

282. "According to one embodiment … a network computer ('NC') system maintains a copy of the OS that cannot be corrupted by ordinary users of the NC system." *Id*., 2:56-59.

283. "Additionally, the NC system may preserve user customizations (e.g., preferences) by maintaining individual, user, storage areas." *Id*., 2:59-61.

284. "When an NC client boots from the network and accesses a stored copy of the OS from an NC server, the user's preferences are dynamically merged with the system environment provided to the NC client." *Id*., 2:62-65.

285. Thus, because "the user's desktop preferences and other customized settings are preserved from session to session and supplied to the NC client as it boots from the network, the user may login to any NC client on the network and have the same user experience." *Id*., 2:65-3:3.

286. Birse's "FIG. 1 is a simplified block diagram that conceptually illustrates an NC system 100 according to one embodiment of the present invention" where "the NC system 100 includes an NC client 150 coupled to an NC server 170

via a data communication link 140, such as … Ethernet link." *Id.*, 4:65-5:8, FIG. 1 (reproduced below).

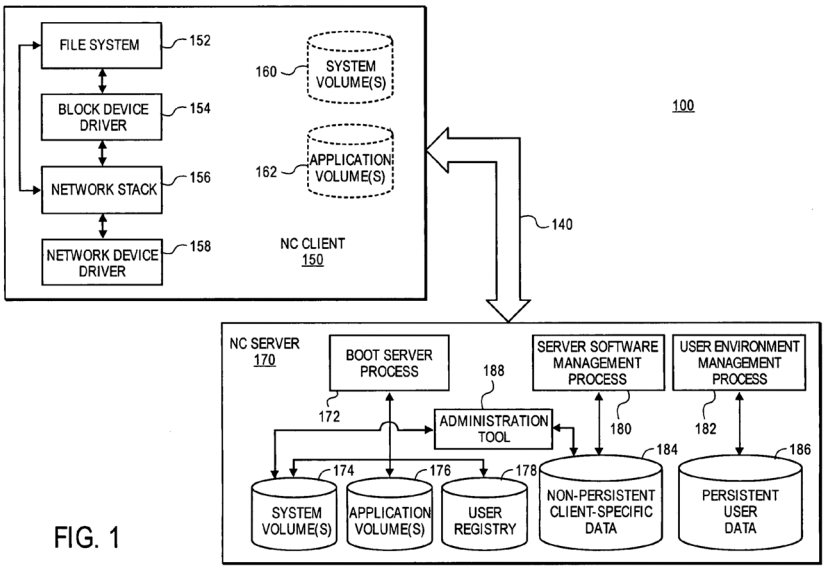


FIG. 1

Birse, Figure 1

287. "The NC server 170 includes one or more system volumes 174, one or more application volumes 176, a user registry 178, non-persistent client-specific data 184, and persistent user data 186." *Id.*, 5:9-29.

288. Moreover, "[t]he system volumes 174 include a protected, read-only, master copy of the operating system software. The application volumes 176 include copies of various application programs for use by the NC client 150" and a "user registry 178 [that] is a database of authorized users, user passwords, NC client hardware addresses, such as Ethernet Media Access Control (MAC) addresses, and NC client network addresses, such as Internet Protocol (IP) addresses." *Id.*

289. Then, "[t]he non-persistent client-specific data 184 represents temporary data storage that is typically preserved only for the duration of a user session on a particular NC client. The persistent user data 186 represents long-term data storage for user information that is desirable to retain between user sessions, such as preferences, browser bookmarks, and other desktop environment customizations." *Id*., 5:9-29.

290. In Birse, "the boot server process 172 manages access to and from the system volumes 174 and the application volumes 176. Additionally, the boot server process 172 performs server-side bootstrapping processing" for example, using the "conventional Bootstrap Protocol (Bootp) server processing that waits for NC clients to appear on the network in accordance with Request For Comments (RFC) 951, … RFC 2132, and uses the standard extensions format." *Id*., 5:46-56.

291. "[T]o facilitate remote NC system administration, such as the installation of new system or application software, the boot server process 172 may serve application and system images that exist in the NC client's folder at the time of bootstrapping as a read-write file." *Id*., 6:4-8.

292. Birse's "server software management process 180 manages access to and from the non-persistent client-specific data 184. For example, at the conclusion of a user session on an NC client, the server software management process 180 may provide data that is to be preserved between user sessions to the user environment

management process 182 and reinitialize the client-specific storage area associated with the NC client for use by the next user." *Id*., 6:15-22.

293. Birse's "user environment management process 182 tracks and maintains the persistent user data 186 to insure that changes the user has made during the current session will be persistent at the next logic. For example, upon termination of a user session, preferences, desktop items, and various other information representing changes made by the user during the user session are copies to a user-specific storage location on the NC server 170." *Id*., 6:35-42

294. Thus, "at the user's next login, from the same or different NC client, the user environment management process 182 will retrieve the data from the corresponding user-specific storage location and return it to the NC client thereby allowing the user to login to any NC client of the NC system 100 and have the same user experience." *Id*., 6:43-48.

295. Birse's "NC client also includes a file system 152, a block device driver 154, a network stack 156, and a network device driver 158, each of which may include hard-wired circuitry or machine-executable instructions or a combination thereof. Furthermore, at least a portion of such hard-wired circuitry and/or machine-executable instructions may be shared between a combination of the file system 152, the block device driver 154, the network stack 156, and the network device driver 158." *Id*., 7:6-14.

296. "The file system 152 represents logical structures and software routines that are used to control access to and from a local storage medium, such as a hard disk system." *Id*., 7:21-23.

297. "The block device driver 154 services file system read and write requests to the system volumes 160." *Id*., 7:35-35.

298. "The network stack 156 represents logical structures and software routines that are used to control access to and from the data communication link 140." *Id*., 7:55-57.

299. Birse discloses an NC client/server system that includes a base/root image (Core System Volume 922) and at least one leaf/incremental snapshot (shadow system volume 930). EX1006 (Birse), 14:24-30, 15:1-7. This is seen in Birse's Figure 9, reproduced below.

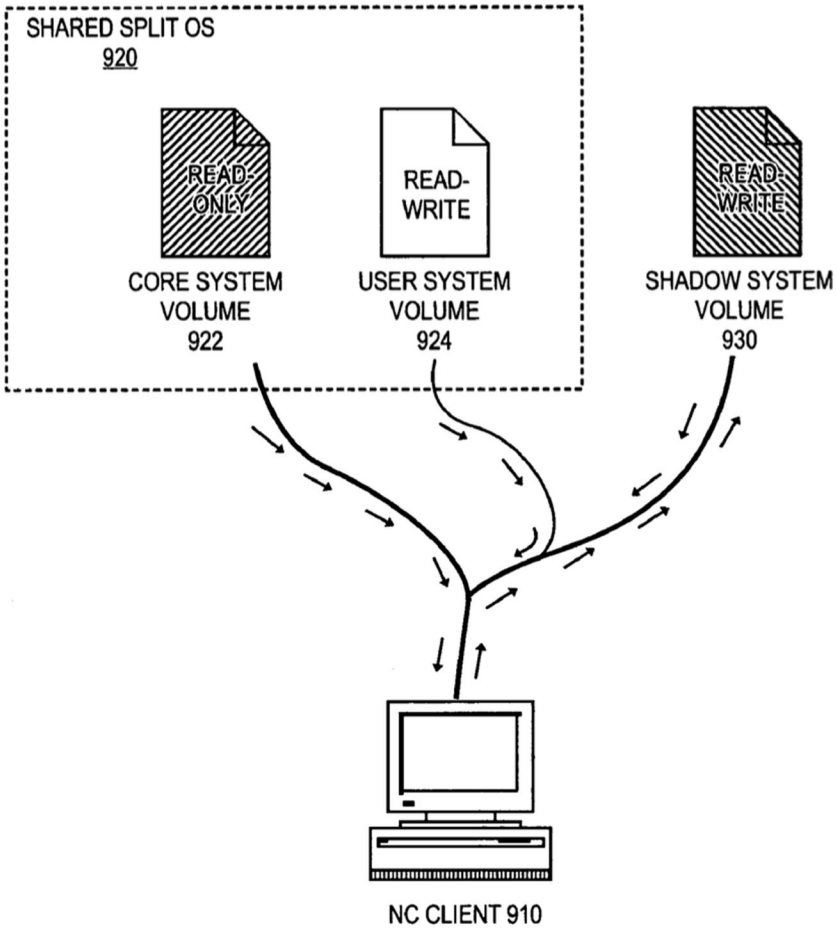


FIG. 9

300. "FIG. 9 is a block diagram that conceptually illustrates NC client interaction with a SplitOS 920 according to one embodiment of the present invention. According to the example depicted, the SplitOS 920 of the NC server 170 contains a read-only core system volume image 922 and a read-write user system volume image 924." EX1006 (Birse), 14:24-30.

301. Moreover, Birse states for Figure 9 that "[a]s indicated by the directional arrows along the connections between the core system volume 922, the user system volume 924, and the shadow system volume 930 and the NC client 910, data may be read from each of the core system volume 922, the user system volume 924, and the shadow system volume 930; however, data may only be written to the shadow system volume 930." EX1006 (Birse), 15:1-7. A POSITA would have understood Birse to manage snapshots as I describe in §VII.E above.

302. In Birse, "the shadow system volume 930 is used by the block device driver 154 of the NC client 150 to implement a 'copy-on-write' storage scheme in which modifications (writes) 60 directed to the user system volume 930 are instead copied to the shadow system volume 930." EX1006 (Birse), 14:57-61. A POSITA would have understood Birse to manage snapshots using standard CoW techniques. *See also* §VII.E above.

303. "The block device driver 154 services file system read and write requests to the system volumes 160. In one embodiment, the block device driver 154 appears to the file system 152 to be a standard hard drive device driver. However, in reality, the block device driver 154 may be configured to service the read and write requests of the file system 152 by accessing the NC server 170." EX1006 (Birse), 7:35-41. A POSITA would have understood Birse to include a (logical/virtual) device driver that resides between file systems and storage devices,

and merges/unifies I/O requests from applications. This functionality is an integral part of snapshot storage systems. *See* §VII.E above.

304. Birse discloses typical operating systems that use typical memory and caches. "The digital processing system 200 includes a processor 252, which may represent one or more processors and may include one or more conventional types of such processors, such as Motorola Power PC processor, an Intel Pentium (or x86) processor, etc. A memory 254 is coupled to the processor 252 by a bus 256. The memory 254 may be a dynamic random access memory (DRAM) and/or may include static RAM (SRAM). The processor may also be coupled to other types of storage areas/memories (e.g., cache, Flash memory, disk, etc.), which could be considered as part of the memory 254 or separate from the memory 254." EX1006 (Birse), 8:10-20.

305. Birse discloses that "the shadow system volume 930 is used by the block device driver 154 of the NC client 150 to implement a 'copy-on-write' storage scheme in which modifications (writes) 60 directed to the user system volume 930 are instead copied to the shadow system volume 930." EX1006 (Birse), 14:57-61. A POSITA would have understood Birse's block driver to implement the functionality of managing snapshots including merging their content, using standard CoW techniques. *See also* §VII.E above.

306. In sum, Birse discloses data structures and methods that a POSITA would have found functionally identical to the '844's UBD, especially in light of the '844 patent's description of what a UBD may be and how it reportedly works (*see* §VIII.C above). Moreover, Birse operates and integrates with caches as discussed above. Lastly, a POSITA would have known that any computer systems, storage system, and more integrate with caches: that is, the software systems (*e.g.*, operating systems, applications) often consult the fast caches first, then the underlying (and typically slower layers), as I detail in §VII.D above.

XI. CLAIM CONSTRUCTION

307. I understand that, for purposes of my analysis in this *inter partes* review proceeding, the terms appearing in the patent claims should be interpreted according to their "ordinary and customary meaning." In determining the ordinary and customary meaning, I understand that the words of a claim are first given their plain meaning that those words would have had to a person of ordinary skill in the art at the time of the alleged invention. I also understand that the structure of the claims, the specification, and file history may be used to better construe a claim. Moreover, I understand that even treatises and dictionaries may be used, albeit under limited circumstances, to determine the meaning attributed by a person of ordinary skill in the art to a claim term.

308. As detailed in my analysis below, the challenged claims are obvious under any reasonable interpretation of the claim language.

309. For purposes of this declaration, I've been asked to assume that the time of the alleged invention is December 30, 2004 (the filing date of U.S. Patent Application 11/026,622, EX1009).

XII. DETAILED EXPLANATION OF GROUNDS

A. Ground 1: Sapuntzakis In View Of Holtzman Renders Claim 1-13 Obvious.

310. I discuss claims 7-13 first, followed by claims 1-6 (which recite similar limitations as claims 7-13).

1. Claim 7

a. [7.0] A method for providing data to a plurality of compute nodes, comprising:

311. Sapuntzakis discloses a method for providing data (capsules) to a plurality of compute nodes (user computers): "Computers and storage in the Collective system act as caches of capsules. As users travel, the Collective can move their capsules to computers close to them, giving users a consistent environment. Capsules could be moved with users as they commute between home and work. Capsules can be duplicated, distributed to many different machines, and updated like any other data; this can form the basis for administering a group of computers." EX1005 (Sapuntzakis), p. 2.

b. [7.1] storing blocks of a root image of said compute nodes on a first storage unit;

312. Sapuntzakis discloses storing blocks of a root image (root disk) of said compute nodes (computers) on a first storage unit (remote COW disks on HCP server of Figure 3): "At the root of the hierarchy is a *root disk*, which is a complete capsule disk." EX1005 (Sapuntzakis), p. 5.

313. "Figure 1 shows an example of a capsule hierarchy illustrating how it may be used in a university. The root capsule contains all the software available to all students." EX1005 (Sapuntzakis), p. 5. Figures 1-3 (reproduced below) show the root COW disk:

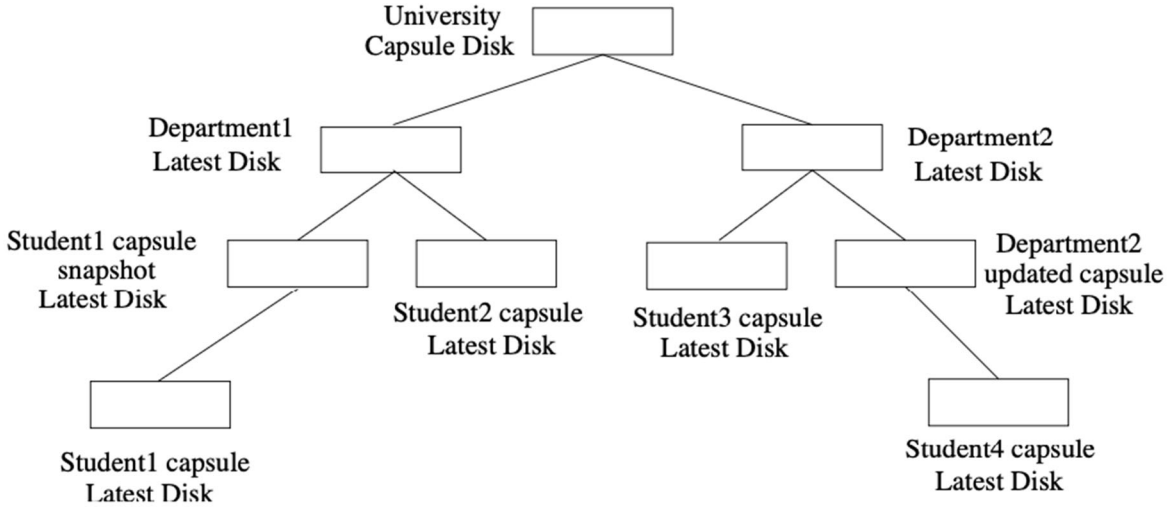


Figure 1: An example capsule hierarchy.

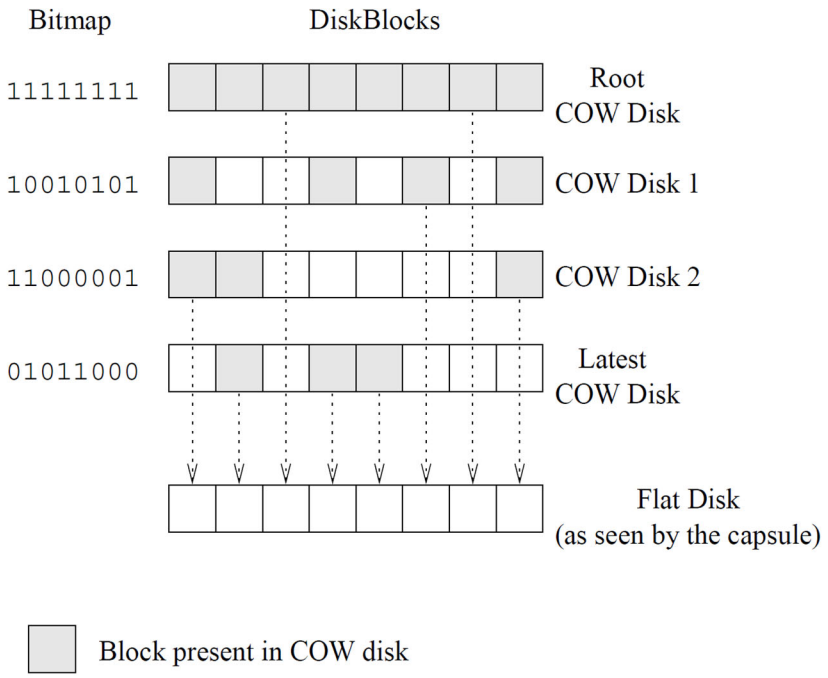


Figure 2: An example capsule disk.

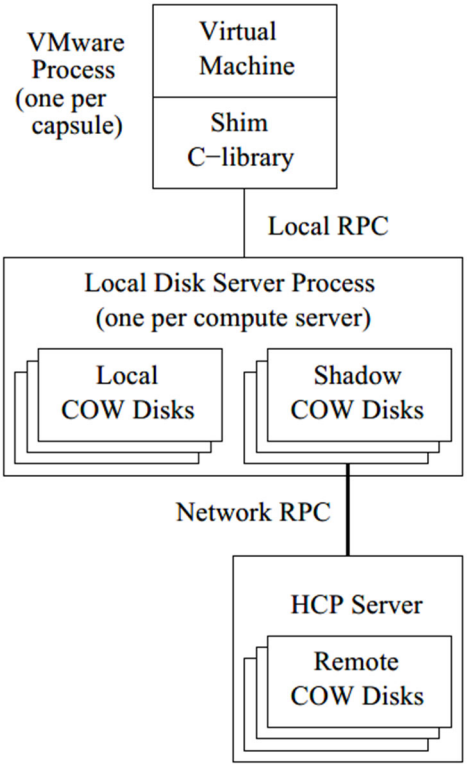


Figure 3: Implementation of capsule disks and demand paging.

PETITIONERS - AMERICAN/SOUTHWEST, Exhibit 1003
Page 125 of 228

314. A POSITA would have understood that the remote COW Disk images of Figure 3 would have been stored on a first storage unit of the HCP server because it is the central server hosting these images.

315. In sum, Sapuntzakis discloses storing blocks of a root image (root disk) of said compute nodes (computers) on a first storage unit (remote COW disks on HCP server of Figure 3).

c. [7.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

316. Sapuntzakis discloses storing leaf images (the local COW disk images of Figures 2 and 3) for respective compute nodes on respective second storage units (attached either to the HCP server or the client computer), said leaf images including only additional data blocks not previously contained in said root image (write requests for new data) and changes made by respective compute nodes to the blocks of the root image (write request for changes to operating system data), wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes (only new data and changes are written): "We can store the disks in these capsules efficiently by creating a hierarchy, where each child capsule could be viewed as inheriting from the parent capsule with

the differences in disk state between parent and child captured in a separate copy-on-write (COW) virtual disk." EX1005 (Sapuntzakis), p. 5.

317. The local COW disk images holding the leaf images would have been stored on respective second storage units attached to the (local) client computers: "Writes to a capsule disk are performed by writing the data to the latest COW disk and updating its bitmap file. Reads involve searching the latest COW disk and its ancestor disks in turn until the required block is found. Since the root COW disk contains a copy of all the blocks, the search is guaranteed to terminate." EX1005 (Sapuntzakis), p. 5-6. See FIG. 2, showing how only modified blocks of data (leaf images) are stored in COW Disk 1, COW Disk 2, and so on.

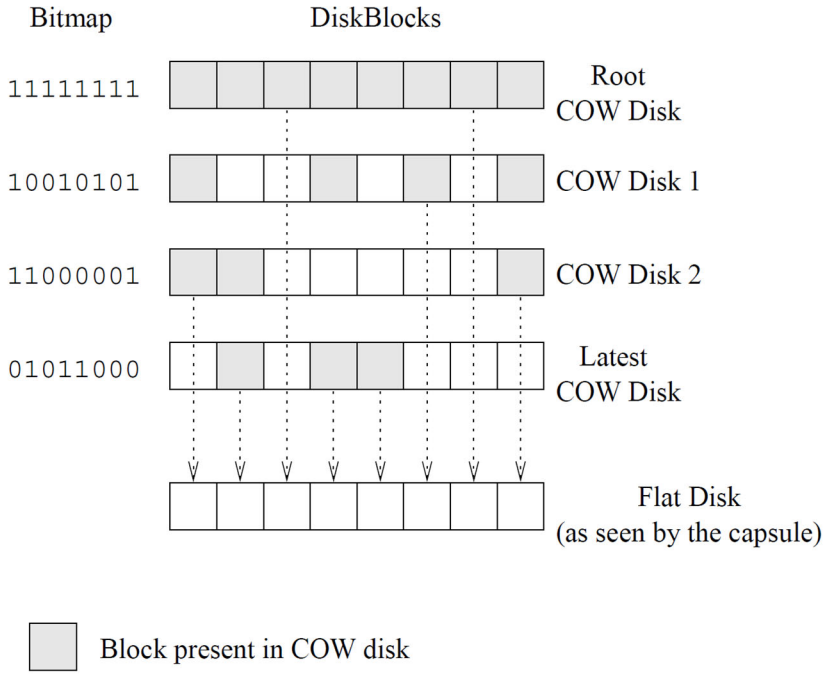


Figure 2: An example capsule disk.

318. "Since the latest COW disk is always local, all writes are local." EX1005 (Sapuntzakis), p. 6.

319. A POSITA would have understood that, because writes to the local (latest) COW disk are described to occur only in these circumstances (when existing blocks are changed or overwritten), the local COW disk will include only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image and will not include blocks of the root image that are unchanged by respective compute nodes. This is standard operation of CoW snapshots. *See* §VII.E above.

320. In sum, Sapuntzakis discloses storing leaf images (the local COW disk images of Figure 3) for respective compute nodes on respective second storage units (attached either to the HCP server or the client computer), said leaf images including only additional data blocks not previously contained in said root image (write requests for new data) and changes made by respective compute nodes to the blocks of the root image (write request for changes to operating system data), wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes (only new data and changes are written).

d. [7.3] caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.

321. The combination of Sapuntzakis and Holzmann discloses caching blocks of said root image (the remote COW disks) that have been accessed by at least one of said compute nodes in a cache memory (the shadow COW disks).

322. Sapuntzakis states: "Each COW disk can either be local or remote. Each remote COW disk has a corresponding local *shadow COW* disk which contains all the locally cached blocks of the remote COW disk." EX1005 (Sapuntzakis), p. 6.

323. "Reads, on the other hand, could either be local or remote. In the case of a remote read, the disk server requests the block from the shadow COW disk. If the block is not cached locally, it is fetched remotely and added to the shadow COW." EX1005 (Sapuntzakis), p. 6.

324. "First, the memory in most systems caches disk blocks." EX1005 (Sapuntzakis), p. 7.

325. Holzmann states: "…[T]he invention [of Holzmann] is directed to a system and method for booting multiple servers using a single operating system image located on a solid state disk in a storage area network." EX1007 (Holzmann), ¶[0002].

326. While referred to as servers in Holzmann, a POSITA would have understood there is no meaningful difference between compute nodes and servers

and these correspond to the compute nodes of the '844 Patent and the HCP servers and clients in Sapuntzakis. *See* §VII.A above. Holzmann's Figure 1 (reproduced below) illustrates this:

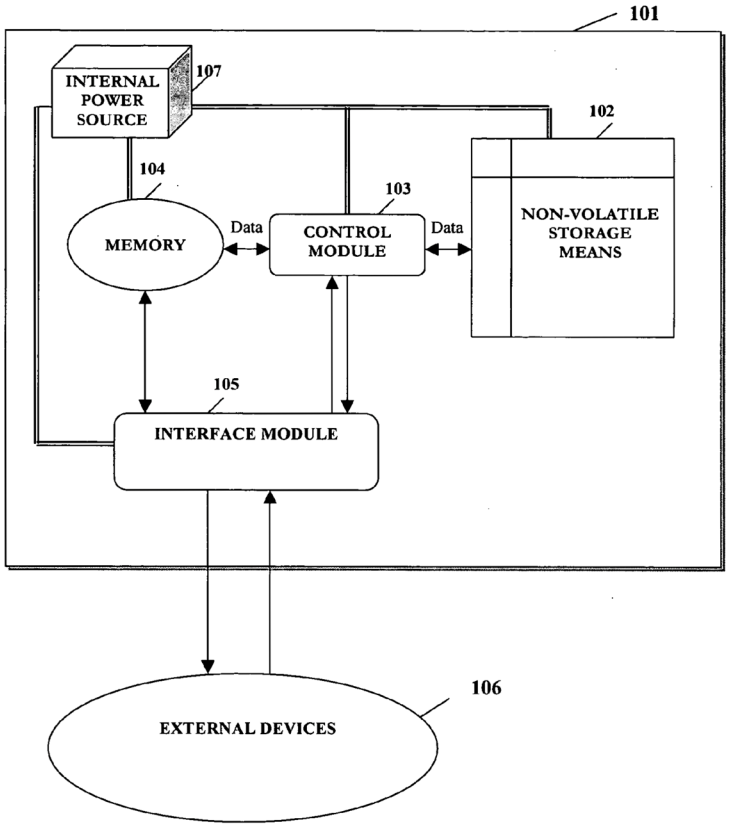


Fig. 1

Holzmann, Figure 1

327. "FIG. 1 is a block diagram illustrating the component parts of the invented system and the functions of each component part. The invented system comprises a solid state disk system 101 having a storage means 102, a control module 103, a memory module 104, an interface module 105 that communicates

with external devices 106, and an internal power source 107." EX1007 (Holzmann), ¶[0017].

328. "The memory module 104 comprises at least one direct-access memory module for holding data currently or recently used by external devices 106. The memory module 104 is more quickly accessible, and performs read and write processing functions more quickly, than non-volatile or disk memory, such as storage means 102. The memory module 104 preferably comprises at least one random-access memory (RAM) module." *Id*., ¶[0020].

329. A POSITA would have recognized this passage as describing a cache memory because RAM is often used as a fast volatile cache and (SSD) storage devices are used as relatively fast non-volatile caches. *See also* §VII.D above.

330. "Upon receiving a boot request, the control module 103 loads the required portion of the operating system image (the portion loaded typically depends on the operating system and its boot process) into memory module 104, … Portions of the cache may be assigned statically for each server or other external devices 106. Alternatively, and preferably, the portions of the cache attributed to each server may be assigned dynamically…" EX1007 (Holzmann), ¶[0021].

331. "The external device 106 requests operating system data through the interface module 105. […T]he control module 103 monitors the request for operating system data, and determines whether the particular operating system data

block requested by the external devices 106 has previously been loaded into the operating system cache. If the data block has been loaded into the cache, and that data block has not been overwritten …, the requested data block is provided from the ….” *Id*., ¶[0024].

332. “If a second server or other external device sends a boot request to control module 103, the module allocates additional space in the cache for a second cache 402…. If additional servers require boot operations, additional cache space 403 can [be] allocated for those servers. By allocating separate cache space for each server or other network resource, the inventive system prevents corruption of data, overwriting of data, and other harmful actions that would lead to resource malfunction.” *Id*., ¶[0033].

333. A POSITA would have understood that, when the cache of Holzmann is used to cache blocks of the read-only shared OS image 660 of Sapuntzakis, there is no need to create separate caches for each client because the read-only image is not subject to “corruption of data, overwriting of data, and other harmful actions that would lead to resource malfunction.” EX1007 (Holzmann), ¶[0033]. That is, read-only data can be easily shared with multiple users and they will all see the same consistent data.

334. Finally, "[i]f the requested data has not been loaded into the cache 501, the control module 103 copies the data from the operating system data stored in memory portion 400 to the cache 501." *Id*., ¶¶[0034-35].

335. In sum, Sapuntzakis and Holzmann, disclose caching blocks of said root image (the remote COW disks) that have been accessed by at least one of said compute nodes in a cache memory (the shadow COW disks).

2. Motivation to Combine Sapuntzakis and Holzmann

336. A POSITA would have been motivated to combine Sapuntzakis and Holzmann for at least the following five reasons.

337. **First**, Sapuntzakis and Holzmann are both concerned with the management of whole operating system (OS) images.

338. Sapuntzakis "shows how to quickly move the state of a running computer across a network, including the state in its disks, memory, CPU registers, and I/O devices." EX1005 (Sapuntzakis), p. 1, Abstract.

339. Similarly, Holzmann "is directed to a system and method for booting multiple servers or other network resources from a single operating system image." EX1007 (Holzmann), Abstract.

340. **Second**, Sapuntzakis and Holzmann are both concerned with the efficient management of these OS images.

341. Sapuntzakis discloses to "have developed techniques to reduce the amount of data sent over the network: copy-on-write disks track just the updates to capsule disks, 'ballooning' zeros unused memory, demand paging fetches only needed blocks, and hashing avoids sending blocks that already exist at the remote end" and further that "experimental results suggest that efficient capsule migration can improve user mobility and system management." EX1005 (Sapuntzakis), Abstract.

342. Holzmann similarly recognizes some of the core elements that improve system efficiency: faster storage devices and shared network storage systems. "SSD systems can access data more quickly and process more read and write requests than conventional disk systems found on most server computers." EX1007 (Holzmann), ¶[0004]. "Booting servers from a SAN offers numerous advantages. The SAN allows the various operating system images to be stored centrally, allowing for efficient loading, monitoring, patching and updating of the operating system. Central storage of operating system images on the SAN also facilitates easier replacement or swapping of server hardware." EX1007 (Holzmann), ¶[0008].

343. **Third**, both Sapuntzakis and Holzmann disclose caching blocks of the root/boot disk images.

344. "Each COW disk can either be local or remote. Each remote COW disk has a corresponding local shadow COW disk which contains all the locally cached

blocks of the remote COW disk." EX1005 (Sapuntzakis), p. 6. "Note that since remote COW disks are immutable, the cached blocks in the shadow COW disks can be re-used for multiple capsules and across suspends and resumes of a single capsule. This is useful since no network traffic is incurred for the cached blocks." EX1005 (Sapuntzakis), p. 7.

345. Although Sapuntzakis discloses caching of disk image blocks, it does not necessarily detail how caching may be applied to each disk image of a separate operating system. A POSITA would still have understood that when Sapuntzakis migrates a plurality of virtual machine images, each image's blocks are cached separately and yet share common data as needed to improve efficiency.

346. "Capsule hierarchies allow efficient usage of disk space by sharing common data among different capsule disks. This also translates to efficient usage of the buffer cache of the host OS, when multiple capsules sharing COW disks simultaneously execute on the same host. And finally, creating a new capsule using COW disks is much faster than copying entire disks." EX1005 (Sapuntzakis), p. 5.

347. Holzmann goes a step further and discloses details of caches for each system/server being booted, and different cache configurations.

348. Holzmann suggests various forms of caches. "Alternatively, the present system may be configured to allow for dynamic allocation of cache resources through the use of a non-associative cache, as depicted in FIG. 5." EX1007

(Holzmann), ¶[0034]. "[S]till more cache configurations can be used in accordance with the invented system, including additional set-associative caches or fully associative caches." EX1007 (Holzmann), ¶[0038].

349. Holzmann discloses specific examples where each server or operating system has its own dedicated cache.

350. "As multiple servers or other external devices 106 are booted, the control module 103 allocates additional space into the cache stored in memory module 104. Portions of the cache may be assigned statically for each server or other external devices 106. Alternatively, and preferably, the portions of the cache attributed to each server may be assigned dynamically, as described further below with reference to FIG. 5." EX1007 (Holzmann), ¶[0021], Fig. 5 (reproduced below).

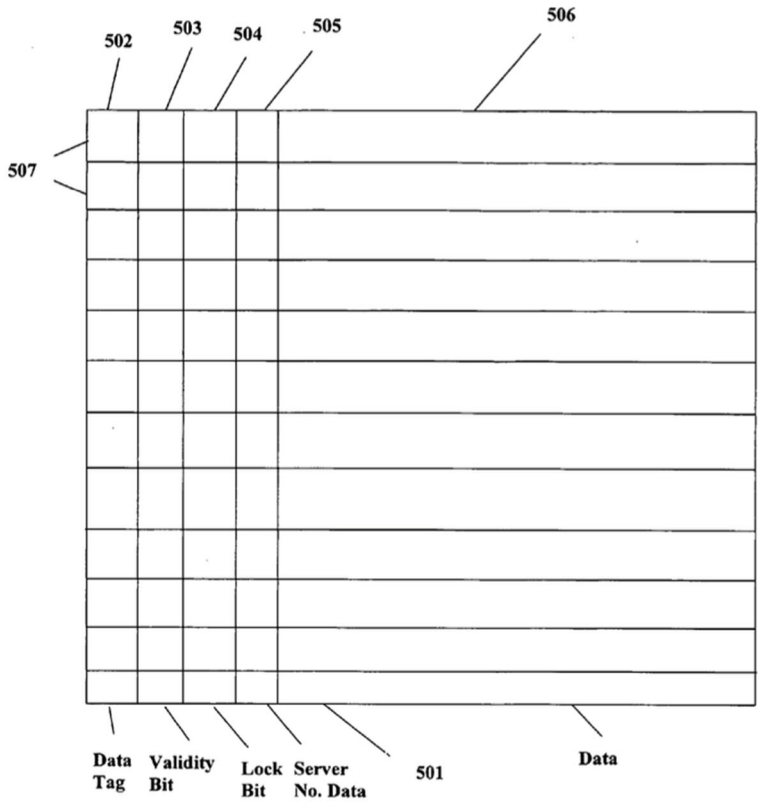


Fig. 5

Holzmann, Figure 5

351. Furthermore, "[b]y allocating separate cache space for each server or other network resource, the inventive system prevents corruption of data, overwriting of data, and other harmful actions that would lead to resource malfunction." EX1007 (Holzmann), ¶[0033].

352. **Fourth**, A POSITA would have been motivated to modify Sapuntzakis with Holzmann's caching techniques. Sapuntzakis discloses caching but a POSITA would have known that there are many different caching techniques with their own pros and cons, as detailed in §VII.D above.

353. Specifically, and as discussed immediately above, Holzmann offers details of how to cache data for each operating system (*e.g.*, Sapuntzakis's virtual machine). Holzmann goes further and discloses several forms of caches: unique caches per server, caches that are statically or dynamically allocated, non-associative caches, set-associative caches, and fully associative ones.

354. **Fifth** and lastly, a POSITA would have known that caching is an age-old technique that has been commonly used in various locations for decades before the '844 Patent. *See* §VII.D above. In addition, caching data in RAM memory is as old as computers themselves. *See* §VII.A above. A POSITA interested in even more details of caching than either Sapuntzakis or Holzmann disclose could easily consult the extensive literature on the subject.

355. Thus, modifying Sapuntzakis with Holzmann's caches would have been simple and would not have required extensive experimentation or impose undue burden, because both already disclose caching. For the same reasons, the combination would have yielded predictable results (because caching is well understood to improve performance and resource utilization). Lastly, a POSITA would have had a high expectation of success because Holzmann does nothing more than describe caching in more detail than Sapuntzakis does.

3. Claim 8

a. [8.1] The method as recited in claim 7 further comprising: receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and

356. For the method as recited in claim 7, see claim limitations [7.1]-[7.3] in this Ground 1 (§XII.A.1 above).

357. The combination of Sapuntzakis and Holzmann discloses receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory (the shadow COW disks). Moreover, this limitation describes nothing more than common caching behavior (finding if something is cached in response to a read/write request). *See* §VII.D above.

358. Sapuntzakis states: "Reads, on the other hand, could either be local or remote. In the case of a remote read, the disk server requests the block from the shadow COW disk. If the block is not cached locally, it is fetched remotely and added to the shadow COW." EX1005 (Sapuntzakis), p. 6.

359. Holzmann states: "the control module 103 monitors the request for operating system data, and determines whether the particular operating system data block requested by the external devices 106 has previously been loaded into the operating system cache. If the data block has been loaded into the cache, and that

data block has not been overwritten subsequent to being loaded into the cache, the requested data block is provided from the cache in memory module 104 to interface module 105 for transmission to the appropriate external device 106." EX1007 (Holzmann), ¶[0024].

360. A POSITA would have understood that, when the cache of Holzmann is used to cache blocks of the read-only shared OS image 660 of Sapuntzakis, there is no need to create separate caches for each client because the read-only image is not subject to "corruption of data, overwriting of data, and other harmful actions that would lead to resource malfunction." EX1007 (Holzmann), ¶[0033]. That is, read-only data can be easily shared with multiple users and they will all see the same consistent data.

361. In sum, Sapuntzakis and Holzmann disclose receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory (the shadow COW disks).

b. [8.2] providing said first portion of said data to said at least one of said compute nodes from said cache memory.

362. The combination of Sapuntzakis and Holzmann discloses providing said first portion of said data to said at least one of said compute nodes from said cache memory (the shadow COW disks). Moreover, this limitation describes

nothing more than common caching behavior (returning a cached item if found). *See* §VII.D above.

363. Sapuntzakis states that "[r]eads, on the other hand, could either be local or remote. In the case of a remote read, the disk server requests the block from the shadow COW disk. If the block is not cached locally, it is fetched remotely and added to the shadow COW." EX1005 (Sapuntzakis), p. 6.

364. Holzmann states that "the control module 103 monitors the request for operating system data, and determines whether the particular operating system data block requested by the external devices 106 has previously been loaded into the operating system cache. If the data block has been loaded into the cache, and that data block has not been overwritten subsequent to being loaded into the cache, the requested data block is provided from the cache in memory module 104 to interface module 105 for transmission to the appropriate external device 106." EX1007 (Holzmann), ¶[0024].

365. A POSITA would have understood from this passage that if the requested blocks are cached, they are served from the cache as is common for caches. *See* §VII.D above.

366. In sum, Sapuntzakis and Holzmann disclose providing said first portion of said data to said at least one of said compute nodes from said cache memory (the shadow COW disks).

4. Claim 9: The method as recited in claim 8 further comprising: updating said cache memory based on said read request.

367. For the method as recited in claim 8, see claim limitations [8.1]-[8.2] in this Ground 1 (§XII.A.3 above).

368. The combination of Sapuntzakis and Holzmann further discloses updating said cache memory (the shadow COW disks) based on said read request. Moreover, this limitation describes nothing more than common caching behavior (updating a cache based on reading a new item, called a "cache admission" policy). *See* §VII.D above.

369. Sapuntzakis states that "[r]eads, on the other hand, could either be local or remote. In the case of a remote read, the disk server requests the block from the shadow COW disk. If the block is not cached locally, it is fetched remotely and added to the shadow COW." EX1005 (Sapuntzakis), p. 6.

370. Holzmann states that "[i]f the requested data has not been loaded into the cache 501, the control module 103 copies the data from the operating system data stored in memory portion 400 to the cache 501." EX1007 (Holzmann), ¶[0035].

371. A POSITA would have understood from this passage that if the requested blocks are not cached, they are loaded from backing store and stored in the cache for subsequent faster reading from the cache. *See* §VII.D above.

372. In sum, Sapuntzakis and Holzmann further discloses updating said cache memory (the shadow COW disks) based on said read request.

5. Claim 10

a. [10.0] The method as recited in claim 9 wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises:

373. For the method as recited in claim 8, see claim [9] in this Ground 1 (§XII.A.4 above).

374. The combination of Sapuntzakis and Holzmann discloses processing a request wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises: caching said second portion in said cache memory. Moreover, this limitation describes nothing more than common caching behavior (requesting a plurality of items from a cache). *See* §VII.D above.

375. Sapuntzakis states that "[r]eads, on the other hand, could either be local or remote. In the case of a remote read, the disk server requests the block from the shadow COW disk. If the block is not cached locally, it is fetched remotely and added to the shadow COW." EX1005 (Sapuntzakis), p. 6.

376. Holzmann states that "the control module 103 monitors the request for operating system data, and determines whether the particular operating system data block requested by the external devices 106 has previously been loaded into the operating system cache. If the data block has been loaded into the cache, and that

data block has not been overwritten subsequent to being loaded into the cache, the requested data block is provided from the cache in memory module 104 to interface module 105 for transmission to the appropriate external device 106." EX1007 (Holzmann), ¶[0024].

377. A POSITA would have thus understood that the combination of Sapuntzakis and Holzmann would allow for some data to not be cached at certain points in time.

378. In sum, Sapuntzakis and Holzmann disclose processing a request wherein a second portion of the data requested is not currently stored in said cache memory.

b. [10.1] caching said second portion in said cache memory; and

379. Sapuntzakis and Holzmann further disclose updating said cache memory (the shadow COW disks) based on said read request. Moreover, this limitation describes nothing more than common caching behavior (updating a cache based on reading a new item, called a "cache admission" policy). *See* §VII.D above.

380. Sapuntzakis states that "[r]eads, on the other hand, could either be local or remote. In the case of a remote read, the disk server requests the block from the shadow COW disk. If the block is not cached locally, it is fetched remotely and added to the shadow COW." EX1005 (Sapuntzakis), p. 6.

381. Holzmann states that "[i]f the requested data has not been loaded into the cache 501, the control module 103 copies the data from the operating system data stored in memory portion 400 to the cache 501." EX1007 (Holzmann), ¶[0035].

382. A POSITA would have understood from this passage that if the requested blocks are not cached, they are loaded from backing store and stored in the cache for subsequent faster reading from the cache. *See* §VII.D above.

383. In sum, Sapuntzakis and Holzmann disclose processing a request wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises: caching said second portion in said cache memory.

c. [10.2] removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value.

384. The combination of Sapuntzakis and Holzmann discloses removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value. Moreover, this limitation describes nothing more than common caching behavior (freeing some items from a cache to make room for new items, called a "cache eviction" policy). *See* §VII.D above.

385. Sapuntzakis states that "[o]n a miss, the cache adds the entry to the bucket, possibly evicting an existing entry. Each entry contains a use count that the cache increments on every hit. When adding an entry to the cache, the hash cache

chooses a fraction of the entries at random from the bucket and replaces the entry with the lowest use count; this evicts the least used and hopefully least useful entry of the group." EX1005 (Sapuntzakis), p. 8.

386. Holzmann states that "[i]f the requested data has not been loaded into the cache 501, the control module 103 copies the data from the operating system data stored in memory portion 400 to the cache 501." EX1007 (Holzmann), ¶[0035].

387. Moreover, "[i]t will be appreciated by those skilled in the art that other cache configurations may be used in accordance with the present invention, such as a set associative cache. EX1007 (Holzmann), ¶[0037]. "It will be understood by those skilled in the art that still more cache configurations can be used in accordance with the invented system, including additional set-associative caches or fully associative caches. EX1007 (Holzmann), ¶[0038].

388. A POSITA would have understood this limitation describes nothing more than common caching behavior (removing some cached items to make room for others, called a "cache eviction" policy). *See* §VII.D above.

389. In sum, Sapuntzakis and Holzmann disclose removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value.

6. Claim 11: The method as recited in claim 7 further comprising: merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments.

390. For the method as recited in claim 7, see claim limitations [7.0]–[7.2] in this Ground 1 (§XII.A.1 above).

391. Sapuntzakis further discloses merging the blocks of said root image (the root COW disk) with the blocks of respective leaf images (the local COW disk and ancestors between it and the root) to create cohesive respective application environments. Moreover, this limitation describes nothing more than common snapshotting behavior (traversing snapshots down to a base image and logically merging all blocks to produce a cohesive image that can be mounted as a file system for applications to use). *See* §VII.E above.

392. "To reduce the start-up time of a capsule, the COW disks corresponding to a capsule disk are read page-by-page on demand, rather than being pre-fetched." EX1005 (Sapuntzakis), p. 6.

393. "Reads involve searching the latest COW disk [the leaf image] and its ancestor disks in turn until the required block is found. Since the root COW disk contains a copy of all the blocks, the search is guaranteed to terminate." EX1005 (Sapuntzakis), p. 5–6.

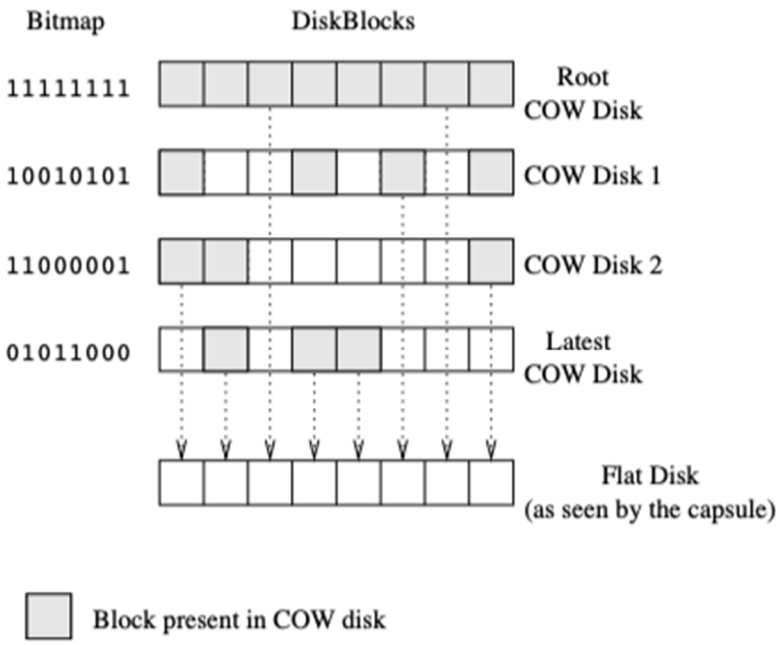


Figure 2: An example capsule disk.

Sapuntzakis, Figure 2

394. "Note that the COW disk hierarchy is not visible to the VMM, or to the OS and applications inside the capsule; all of them see a normal flat disk as illustrated in the figure." EX1005 (Sapuntzakis), p. 6.

395. A POSITA would have understood this to mean that none of the individual COW disks in the hierarchy are visible to the client user; instead, a single cohesive (logical) storage image corresponding to the '844 Patent's merged root and leaf blocks presenting cohesive respective application environments.

396. In sum, Sapuntzakis further discloses merging the blocks of said root image (the root COW disk) with the blocks of respective leaf images (the local COW

disk and ancestors between it and the root) to create cohesive respective application environments.

7. Claim 12: The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory.

397. For the method as recited in claim 11, see claim [11] in this Ground 1 (§XII.A.6 above).

398. Sapuntzakis discloses said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit (the root COW disks), respective second storage units (the local COW disks), and said cache memory (the shadow COW disks).

399. Sapuntzakis states that "[t]he COW disk implementation interfaces with GSX Server through a shim library that sits between GSX Server and the C library. The shim library intercepts GSX Server's I/O requests to disk image files in VMware's 'plain disk' format, and redirects them to a local disk server." EX1005 (Sapuntzakis), p. 6.

400. "The local disk server translates these requests to COW disk requests, and executes the I/O operations against the COW disks." EX1005 (Sapuntzakis), p. 6.

401. "The implementation of the capsule disk system, including demand paging, is shown in Figure 3. The shim library intercepts all of VMware's accesses

to plain disks and forwards them to a disk server on the local machine. The disk server performs a translation from a plain disk access to the corresponding access on the COW disks of the capsule." EX1005 (Sapuntzakis), p. 6.

402. A POSITA would have understood the references to the COW disks in these passages refer to all three of: the root (base image) COW disks, the local (leaf) COW disks, and the shadow (cache) COW disks. Moreover, a POSITA would have understood that even without the presence of COW snapshots, operating systems provide a cohesive view of data that logically merges what is in the cache and what is not cached. *See* §VII.D above.

403. In sum, Sapuntzakis discloses said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit (the root COW disks), respective second storage units (the local COW disks), and said cache memory (the shadow COW disks).

8. Claim 13: The method as recited in claim 7 wherein said first storage unit is read-only.

404. For the method as recited in claim 7, see claim limitations [7.0]-[7.3] in this Ground 1 (§XII.A.1 above).

405. Sapuntzakis further discloses the first storage unit (storing root disk images and possibly certain intermediate COW disks) being read-only.

406. "At the root of the hierarchy is a root disk, which is a complete capsule disk. All other nodes represent a COW disk. Each path of COW disks originating

from the root in the capsule hierarchy represents a capsule disk; the COW disk at the end of the path for a capsule disk is its latest disk. We cannot directly run a capsule whose latest disk is not a leaf of the hierarchy. We must first derive a new child capsule by adding a new child disk to the latest disk and all updates are made to the new disk. Thus, once capsules have children, they become immutable; this property simplifies the caching of capsules." EX1005 (Sapuntzakis), p. 6 (describing Figure 1).

407. A POSITA would have understood that a working operating system has to be able to write some state (*e.g.*, boot and login records, application activity, swap space, and more), and capsules with children are immutable and cannot be run directly because they are read-only.

408. In sum, Sapuntzakis discloses the first storage unit (storing root disk images and possibly certain intermediate COW disks) being read-only.

9. Claim 1

a. [1.0] A system for providing data to a plurality of compute nodes, the system comprising:

409. *See* claim limitation [7.0] in this Ground 1 (§XII.A.1.a above).

b. [1.1] a first storage unit configured to store blocks of a root image of said compute nodes;

410. *See* claim limitation [7.1] in this Ground 1 (§XII.A.1.b above).

c. [1.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

411. *See* claim limitation [7.2] in this Ground 1 (§XII.A.1.c above).

d. [1.3] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes.

412. *See* claim limitation [7.3] in this Ground 1 (§XII.A.1.d above).

10. Claim 2: The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value.

413. *See* claim limitations [1.0]-[1.3] in this Ground 1 (§XII.A.1 above).

414. The combination of Sapuntzakis and Holzmann discloses removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value. Sapuntzakis and Holzmann further disclose said cache being configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value. Moreover, this limitation describes nothing more than common caching behavior (freeing some items from a cache whose size is by definition limited, to make room for new items, called a "cache eviction" policy). *See* §VII.D above.

415. Sapuntzakis states that "[o]n a miss, the cache adds the entry to the bucket, possibly evicting an existing entry. Each entry contains a use count that the cache increments on every hit. When adding an entry to the cache, the hash cache chooses a fraction of the entries at random from the bucket and replaces the entry with the lowest use count; this evicts the least used and hopefully least useful entry of the group." EX1005 (Sapuntzakis), p. 8.

416. Holzmann states that "[i]f the requested data has not been loaded into the cache 501, the control module 103 copies the data from the operating system data stored in memory portion 400 to the cache 501." EX1007 (Holzmann), ¶[0035].

417. Moreover, "[i]t will be appreciated by those skilled in the art that other cache configurations may be used in accordance with the present invention, such as a set associative cache." EX1007 (Holzmann), ¶[0037]. "It will be understood by those skilled in the art that still more cache configurations can be used in accordance with the invented system, including additional set-associative caches or fully associative caches." EX1007 (Holzmann), ¶[0038].

418. A POSITA would have understood this limitation describes nothing more than common caching behavior (cache management policies, where caches naturally have limited room and must use thresholds to decide what to keep/evict). *See* §VII.D above (especially LRU and LFU cache-management policies).

419. In sum, Sapuntzakis and Holzmann further disclose said cache being configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value.

11. Claim 3: The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance.

420. The combination of Sapuntzakis and Holzmann discloses the system as recited in claim 1. *See* claim limitations [1.0]-[1.3] in this Ground 1 (§XII.A.9 above).

421. A POSITA would have understood that much of Sapuntzakis is focused on transferring data over comparatively slow connections, such as DSL or even dialup modems. EX1005 (Sapuntzakis), p. 2 ("An inactive capsule can contain gigabytes of disk storage, whereas an active capsule can include hundreds of megabytes of memory data, as well as internal machine registers and I/O device states. Copying a gigabyte capsule over a standard 384 kbps DSL link would take 6 hours! Clearly, a straightforward implementation that copies the entire capsule before starting its computation would take too long"). In settings where communication speed between server and user computers was not the dominating concern (*i.e.*, for students, faculty and administrators working in offices, labs or dorms on the university network), it could be advantageous to store the "local" COW disks (the leaf images) on the same or similar servers as the root images.

422. The same is true of the cache "shadow" COW disks. Indeed, Sapuntzakis recognizes that "[w]e expect to find identical blocks of data between disk images and memories, even across different users' capsules[' caches]." EX1005 (Sapuntzakis), p. 8. Thus, a shared cache in a centralized storage appliance would be advantageous.

423. Holzmann discloses a cache mechanism for server drives. "The control module 103 then allocates a sub-portion of the memory portion to be used as a cache for the operating system. As multiple servers or other external devices 106 are booted, the control module 103 allocates additional space into the cache stored in memory module 104. Portions of the cache may be assigned statically for each server or other external devices 106. Alternatively, and preferably, the portions of the cache attributed to each server may be assigned dynamically, as described further below with reference to FIG. 5." EX1007 (Holzmann), ¶[0021].

424. A POSITA would have understood that this would also improve the cost/benefit of the use of high-speed solid state, persistent disk-centered memory caches (of the sort present in certain embodiments of Holzmann). For example, a POSITA would have known that the storage systems created by Netapp are all single storage appliances; each such appliance supports multiple volumes, multiple base images, multiple snapshots, and multiple caches (both volatile and non-volatile), as I discuss in §VII.E above.

425. In sum, Sapuntzakis and Holzmann further disclose said first storage unit, said second storage units, and said cache being contained within a single storage appliance.

12. Claim 4: The system as recited in claim 1 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.

426. *See* claim limitations [1.0]-[1.3] (§XII.A.9 above) and [12] (§XII.A.7 above) in this Ground 1.

427. The term Union Block Device is informally defined in the '844 Patent by way of a set of tasks it performs. *See* §VIII.C above. A POSITA would have recognized that such system components as necessary to provide such features, taken together, embody the Union Block Device of this claim. For example, a POSITA would have known that the storage systems created by Netapp are all single storage appliances; each such appliance supports multiple volumes, multiple base images, multiple snapshots, and multiple caches (both volatile and non-volatile). A POSITA would have understood each volume in a Netapp or similar appliance to be the claimed UBD. *See* §VII.E above.

428. In sum, Sapuntzakis discloses a plurality of union block devices configured to interface between respective compute nodes and said first storage unit,

respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image (the root COW disk) with the blocks of respective leaf images (the local COW disk and ancestors between it and the root).

13. Claim 5: The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.

429. *See* claims [4] (§XII.A.12 above) and [12] (§XII.A.7 above) in this Ground 1.

14. Claim 6: The system as recited in claim 1 wherein said first storage unit is read-only.

430. *See* claim limitations [1.0]-[1.3] (§XII.A.9 above) and [13] (§XII.A.8 above) in this Ground 1.

B. Ground 2: Sapuntzakis In View Of Federwisch Renders Claim 14-27 Obvious.

1. Claim 14

a. [14.0] A system for indexing file systems for a plurality of compute nodes, the system comprising:

431. *See* claim [7.0] in Ground 1 above (§XII.A.1.a above).

b. [14.1] a first storage unit configured to store blocks of a root image of said compute nodes;

432. *See* claim [7.1] in Ground 1 above (§XII.A.1.b above).

c. [14.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

433. *See* claim limitation [7.2] in Ground 1 above (§XII.A.1.c above).

d. [14.3]-[14.4] a plurality of union block devices corresponding to said compute nodes, said union block devices configured to interface between said compute nodes and said first and second storage units to distribute said file systems to said compute nodes, wherein said union block devices are configured to create said file systems by merging the blocks of said root image stored on the first storage unit with the blocks of respective leaf images stored on respective second storage units, and wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.

434. The combination of Sapuntzakis and Federwisch further discloses at least one of said compute nodes being configured to index said root image and provide the indexing results to another of said compute nodes.

435. Federwisch discloses a "system and method for remote asynchronous replication or mirroring of changes" between "a source file system snapshot" and "a destination replica file system." EX1008 (Federwisch), Abstract.

436. Federwisch compares two snapshots "using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file system, which

identifies changed blocks in the respective snapshot files based upon differences in volume block numbers identified in a scan of the logical file block index of each snapshot." EX1008 (Federwisch), Abstract. Thus, every snapshot in a volume has a "logical file block index."

437. A POSITA would have understood that the scanner of Federwisch could be implemented on any of the compute nodes of Sapuntzakis, as part of the Union Block Device. A POSITA would have further understood that such scanner could be used to index the root image and to provide the index results to other compute nodes, because such indexing and sharing of data over networks was well known. *See* §VII.C above and §VII.F above.

438. Federwisch compares two snapshots "using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file system, which identifies changed blocks in the respective snapshot files based upon differences in volume block numbers identified in a scan of the logical file block index of each snapshot." EX1008 (Federwisch), Abstract. Thus, every snapshot in a volume has a "logical file block index."

439. Sapuntzakis discloses the remaining limitations of this term. *See* claim [4] in Ground 1 above (§XII.A.12 above).

440. As an initial matter, a POSITA would have known that indexing is an old and common technique used well before 2004. *See* §VII.F above.

441. A POSITA would have been motivated to combine Sapuntzakis and Federwisch for the reasons discussed in §XII.B.2 below.

442. Sapuntzakis discloses that data is indexed but does not provide a lot of details of its indexing. In Sapuntzakis, "[t]he cache is implemented using a hash table whose size is fixed at creation. We use the first several bits of the hash key to index into the table." EX1005 (Sapuntzakis), p. 7. Sapuntzakis's data structure is a hash table that indexes cached entries for faster access. "The hash key indexes into a bucket of entries, currently a memory page in size. On a lookup, the cache does a linear search of the entries in a bucket to check whether one of them matches the hash." EX1005 (Sapuntzakis), p. 7.

443. A POSITA interested in building a system based on Sapuntzakis would have been motivated to seek out additional details on indexing storage systems, files, and caches thereof. This is particularly so because indexing of data items was a well-known technique. *See* §VII.F above. Such a POSITA would have found Federwisch relevant, which discloses greater details of indexing. *See* §X.C above.

444. Federwisch uses common indexing to better manage cached data. "If the information is not in memory, the file system layer 450 indexes into the inode file using the inode number to access an appropriate entry and retrieve a volume block number." EX1008 (Federwisch), ¶[0053].

445. Federwisch consults its indexes to identify changed and unchanged blocks. "The existence of a changed volume block number at a given offset in the index between the base snapshot inode file and incremental snapshot inode file generally indicates that one or more of the underlying inodes and files to which the inodes point have been changed." EX1008 (Federwisch), ¶[0071].

446. Federwisch scans these indexes to identify unchanged blocks. "A scanner 820 searches the index for changed base/incremental inode file snapshot blocks, comparing volume block numbers or another identifier. … Hence, scanned blocks at the same index in both snapshots can be efficiently bypassed, reducing the scan time." EX1008 (Federwisch), ¶[0072].

447. A POSITA would have understood that indexing as described in Federwisch could be used in the system disclosed in Sapuntzakis. Modifying Sapuntzakis to include Federwisch's indexing would have been simple and would not have required extensive experimentation or impose undue burden, because—most likely—Sapuntzakis already indexed all of their data objects. A POSITA would not have expected Sapuntzakis to provide details of a concept as basic as indexing, one taught to first year computer science students

448. In sum, Sapuntzakis and Federwisch further disclose at least one of said compute nodes being configured to index said root image and provide the indexing results to another of said compute nodes. Sapuntzakis discloses that data is indexed.

2. Motivation to Combine Sapuntzakis with Federwisch

449. A POSITA would have been motivated to combine Sapuntzakis and Federwisch for at least the following six reasons.

450. **First**, both operate at and recognize disk blocks at the block layer. A POSITA having reviewed Sapuntzakis would have naturally found other references in the same field of operating on block layer storage systems.

451. For example, Sapuntzakis discloses to "have developed techniques to reduce the amount of data sent over the network: copy-on-write disks track just the updates to capsule disks, 'ballooning' zeros unused memory, demand paging fetches only needed blocks, and hashing avoids sending blocks that already exist at the remote end" and further that "experimental results suggest that efficient capsule migration can improve user mobility and system management." EX1005 (Sapuntzakis), Abstract.

452. Similarly, Federwisch discloses "[a] system and method for remote asynchronous replication or mirroring of changes in a source file system snapshot in a destination replica file system using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file system, which identifies changed blocks in the respective snapshot files based upon differences in volume block numbers identified in a scan of the logical file block index of each snapshot." EX1008 (Federwisch), Abstract.

453. **Second**, both Sapuntzakis and Federwisch disclose and use CoW snapshots. A POSITA would have known that CoW techniques enable snapshots. *See* §VII.E above.

454. *Compare* Sapuntzakis, p. 12 ("The amount of data in the COW disk of each snapshot varied depending on the amount of disk traffic that Winstone generated during that snapshot execution. The large size of the snapshot 2 COW disk is due to Winstone copying a good deal of data at the beginning of the benchmark. The size of the COW disks of all the other snapshots range from 2 to 22 MB after gzip, and can be transferred over completely under about 8 minutes.") *with* Federwisch, Abstract ("A system and method for remote asynchronous replication or mirroring of changes in a source file system snapshot in a destination replica file system using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file system, which identifies changed blocks in the respective snapshot files based upon differences in volume block numbers identified in a scan of the logical file block index of each snapshot.").

455. **Third**, while both operate at the block layer, they also recognize that file systems interoperate and work atop the block layer.

456. For example, Sapuntzakis discloses virtual hard disks are formatted with a file system. "Virtual hard disks are especially powerful. The disks can be backed not just by raw disk devices but by files in the host OS's file system. The

file systems abilities to easily name, create, grow, and shrink storage greatly simplify the management of virtual hard disks." EX1005 (Sapuntzakis), p. 3.

457. Similarly, Federwisch discloses "[a] system and method for remote asynchronous replication or mirroring of changes in a source file system snapshot in a destination replica file system using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file system, which identifies changed blocks in the respective snapshot files based upon differences in volume block numbers identified in a scan of the logical file block index of each snapshot." EX1008 (Federwisch), Abstract.

458. **Fourth**, as discussed in §XII.B.2 above, Sapuntzakis discloses caching blocks of the root/boot disk images. Being such a common technique, it is no surprise that Federwisch also discloses at least two forms of caches: volatile and non-volatile. *See also* §VII.A above.

459. Federwisch caches data in volatile RAM memory caches. "The snapshot is stored on-disk along with the active file system, and is called into the buffer cache of the filer memory as requested by the storage operating system 400 or snapshot mirror application 490 as described further below." EX1008 (Federwisch), ¶[0058]. Moreover, "[t]he inode files may each be loaded into the buffer cache of the source file server memory from the on-disk versions thereof using

conventional disk access processes as directed by the storage operating system snapshot manager (480 in Fig. 4)." EX1008 (Federwisch), ¶[0070].

460. Federwisch also caches data in non-volatile, batter-backed RAM (NVRAM). "The NVRAM is filled, in parallel with the buffer cache, after each client request is completed, but before the result of the request is returned to the requesting client." EX1008 (Federwisch), ¶[0048]. *See also* §VII.E above.

461. **Fifth**, a POSITA would have been motivated to modify Sapuntzakis with Federwisch's indexing.

462. Sapuntzakis discloses that data is indexed but does not provide a lot of details of its indexing. In Sapuntzakis, "[t]he cache is implemented using a hash table whose size is fixed at creation. We use the first several bits of the hash key to index into the table." EX1005 (Sapuntzakis), p. 7. Sapuntzakis's data structure is a hash table that indexes cached entries for faster access. "The hash key indexes into a bucket of entries, currently a memory page in size. On a lookup, the cache does a linear search of the entries in a bucket to check whether one of them matches the hash." EX1005 (Sapuntzakis), p. 7.

463. A POSITA interested in building a system based on Sapuntzakis would have been motivated to seek out additional details on indexing storage systems, files, and caches thereof. This is particularly so because indexing of data items was a

well-known technique. *See* VII.F above. Such a POSITA would have found Federwisch relevant, which discloses greater details of indexing. *See* §X.C above.

464. Federwisch compares two snapshots "using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file system, which identifies changed blocks in the respective snapshot files based upon differences in volume block numbers identified in a scan of the logical file block index of each snapshot." EX1008 (Federwisch), Abstract. Thus, every snapshot in Federwisch's volumes has a "logical file block index."

465. Federwisch uses common indexing to better manage cached data. "If the information is not in memory, the file system layer 450 indexes into the inode file using the inode number to access an appropriate entry and retrieve a volume block number." EX1008 (Federwisch), ¶[0053].

466. Federwisch consults its indexes to identify changed and unchanged blocks. "The existence of a changed volume block number at a given offset in the index between the base snapshot inode file and incremental snapshot inode file generally indicates that one or more of the underlying inodes and files to which the inodes point have been changed." EX1008 (Federwisch), ¶[0071].

467. Federwisch scans these indexes to identify unchanged blocks. "A scanner 820 searches the index for changed base/incremental inode file snapshot blocks, comparing volume block numbers or another identifier. … Hence, scanned

blocks at the same index in both snapshots can be efficiently bypassed, reducing the scan time." EX1008 (Federwisch), ¶[0072].

468. Federwisch even discloses a form of re-indexing of incremental snapshots (which the '844 Patent calls "leaf images"). In one example, "the Destination Snapshot A (2002) is now prepared to transfer changes so as to establish a mirror in Destination Snapshot B (2004). … In this case, both the new source and the new destination system have their own inode maps A and B (2012 and 2014), indexed off the old source 2001, and referencing the inodes in their respective trees. … It then treats the new source as the appropriate map index for the new associated inode map 2018. In the associated map, it stores the new source generation number for the new source index inode number, with each index entry also associated with/mapped to the new destination inode/generation number extracted from the inode map B (2014)." EX1008 (Federwisch), ¶[0137]. *See also* §X.C above.

469. **Sixth** and lastly, a POSITA would have known that indexing of any data is an age-old technique long utilized in computer systems. *See* §VII.F above. Just as a library catalog provides an efficient way of locating books on shelves, so do computers create, manage, and update indexes of many data types (files, disk blocks, images, virtual machines, users, networks, and many more).

470. Thus, modifying Sapuntzakis to include Federwisch's indexing would have been simple and would not have required extensive experimentation or impose undue burden, because—most likely—Sapuntzakis already indexed all of their data objects.

471. For the same reasons, the combination of Sapuntzakis and Federwisch would have yielded predictable results (because indexes are well understood to structure and locate data objects efficiently). Lastly, a POSITA would have had a high expectation of success because indexing techniques have been well understood for decades prior to 2004.

3. Claim 15: The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance.

472. *See* claim limitations [14.0]-[14.4] (§XII.B.1 above) in this Ground 2, and claim [3] (§XII.A.11 above) in Ground 1 above.

4. Claim 16: The system as recited in claim 14 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit and respective second storage units, said union block devices configured to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.

473. *See* claim limitation [14.0]-[14.4] (§XII.B.1 above) in this Ground 2, and claim [4] (§XII.A.12 above) in Ground 1 above.

5. Claim 17: The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.

474. *See* claim [16] (§XII.B.4 above) in this Ground 2, and claim [5] (§XII.A.13 above) in Ground 1 above.

6. Claim 18: The system as recited in claim 14 wherein said first storage unit is read-only.

475. *See* claim limitations [14.0]-[14.4] (§XII.B.1 above) in this Ground 2, and claim [13] (§XII.A.8 above) in Ground 1 above.

7. Claim 19

a. [19.0] A method for indexing file systems for a plurality of compute nodes, comprising:

476. *See* claim limitation [14.0] in this Ground 2 (§XII.B.1.a above).

b. [19.1] storing blocks of a root image of said compute nodes on a first storage unit;

477. *See* claim limitation [14.1] in this Ground 2 (§XII.B.1.b above).

c. [19.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images for respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes;

478. *See* claim limitation [14.2] in this Ground 2 (§XII.B.1.c above).

d. [19.3] merging the blocks of said root image with the blocks of respective leaf images stored on respective second storage units to create respective file systems for respective compute nodes;

479. *See* claim [11] in Ground 1 above (§XII.A.6 above).

e. [19.4] receiving indexing results pertaining to said root image from one of said compute nodes; and

480. *See* claim limitation [14.4] in this Ground 2 (§XII.B.1.d above).

f. [19.4] providing said indexing results to the others of said compute nodes.

481. *See* claim limitation [14.4] in this Ground 2 (§XII.B.1.d above).

8. Claim 20: The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit.

482. *See* claim limitation [14.4] (§XII.B.1.d above) in this Ground 2.

483. As an initial matter, a POSITA would have known that sharing storage and files across devices, say over the network, was well known well before 2004. (Indeed, all of the prior art references, as well as the '844 Patent, rely on central shared storage for various aspects of their respective inventions.) This would have made any data (index or otherwise) conveniently accessible over network storage volumes. *See* §VII.C above.

484. A POSITA would have been motivated to combine Sapuntzakis and Federwisch for the reasons discussed in §XII.B.2 above.

485. *See* claim limitations [19.0]-[19.4] in this Ground 2 (§XII.B.7 above).

486. Federwisch recognizes the usefulness of sharing files and data in a format-agnostic manner to other systems. "More generally, the format should also be independent of the underlying network protocol or device (in the case of a tape or local disk/non-volatile storage) protocol and file system—that is, the information is system 'agnostic,' and not bound to a particular operating system software, thereby allowing source and destination systems of different vendors to share the information. The format should, thus, be completely self-describing requiring no information outside the data stream." EX1008 (Federwisch), ¶[0086].

487. Being a network appliance, Federwisch supports the ability to share files, snapshots, and whole volumes across the network (*e.g.*, using NFS or CIFS). This sharing is called "an export." "The destination file system is exported read-only the user." *Id.*, ¶[0015]. This first snapshot may be an exported snapshot of the replicated snapshot on the destination. *Id.*, ¶[0120].

488. In sum, Sapuntzakis and Federwisch further disclose storing said indexing results on a shared storage unit.

9. Claim 21: The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the compute nodes and said first storage unit and respective second storage units.

489. *See* claim limitations [19.0]-[19.4] (§XII.B.7 above) in this Ground 2, and claim [12] (§XII.A.7 above) in Ground 1 above.

10. Claim 22: The method as recited in claim 19 wherein said first storage unit is read-only.

490. *See* claim limitations [19.0]-[19.4] (§XII.B.7 above) in this Ground 2, and claim [13] (§XII.A.8 above) in Ground 1 above.

11. Claim 23

a. [23.0] A computer-readable storage medium having instructions stored thereon that, in response to execution by at least one computing device, cause the at least one computing device to:

491. *See* claim [19.0] in this Ground 2 (§XII.B.7.a above).

b. [23.1]-[23.3] receive data blocks of a file system, said data blocks comprising a root image portion and leaf image portion, said leaf image portion comprising only additional data blocks not previously contained in said root image portion and changes made by said first compute node to the blocks of said root image, wherein said leaf image portion does not include blocks of said root image that are unchanged by said first compute node, wherein said file system is the result of merging said root image portion and said leaf image portion together at the block-level;

492. *See* claim limitations [19.2] (§XII.B.7.c above) and [19.3] (§XII.B.7.d above) in this Ground 2.

c. [23.4] index said root image portion; and

493. *See* claim limitations [19.4] (§XII.B.7.e above) in this Ground 2.

d. [23.4] provide the results of said indexing to a second compute node, wherein said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node.

494. *See* claim limitations [19.4] (§XII.B.7.f above) and [14.4] (§XII.B.1.d above) in this Ground 2.

12. Claim 24: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to store said results of said indexing on a shared storage unit accessible by said second compute node.

495. *See* claim limitations [23.0]-[23.4] (§XII.B.11 above) and [20] (§XII.B.8 above) in this Ground 2.

13. Claim 25: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion.

496. *See* claim [14.4] (§XII.B.1.d above) in this Ground 2.

497. Moreover, a POSITA would have known that indexing is an old and common technique used well before 2004. Specifically, a POSITA would have understood that when some data changes on a disk or file system, it is inefficient to scan and re-index an entire system, only to reprocess unchanged data; instead, it is more efficient to index only files that have changed. This is easily achieved because storage and file systems include timestamps that denote when the files/data were last modified. *See* §VII.B above and §VII.F above.

498. A POSITA would have been motivated to combine Sapuntzakis and Federwisch for the reasons discussed in §XII.B.2 above.

499. *See* claim limitations [23.0]-[23.4] in this Ground 2 (§XII.B.11 above).

500. The combination of Sapuntzakis and Federwisch further discloses the instructions further causing the at least one computing device to index said leaf image portion.

501. Federwisch compares the indexes to file's block numbers between two snapshots to identify whether data has been changed between the snapshots (as would be accomplished using the CoW techniques discussed in §VII.E above).

502. "The existence of a changed volume block number at a given offset in the index between the base snapshot inode file and incremental snapshot inode file generally indicates that one or more of the underlying inodes and files to which the inodes point have been changed." EX1008 (Federwisch), ¶[0071].

503. Federwisch scans these indexes to identify unchanged blocks. "A scanner 820 searches the index for changed base/incremental inode file snapshot blocks, comparing volume block numbers or another identifier. … Hence, scanned blocks at the same index in both snapshots can be efficiently bypassed, reducing the scan time." EX1008 (Federwisch), ¶[0072].

504. A POSITA would have understood these disclosures to mean that in Federwisch, each snapshot (corresponding to the '844 Patent's "leaf image") has its

own index of file blocks. "More specifically, each entry in the inode map 1400 contains an entry for each inode on the source snapshot. Each inode entry 1402 in the map is indexed and accessed via the source inode number (1404)." EX1008 (Federwisch), ¶[0109].

505. In sum, Sapuntzakis and Federwisch further disclose the instructions further causing the at least one computing device to index said leaf image portion.

14. Claim 26: The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.

506. *See* claim [14.4] (§XII.B.1.d above) in this Ground 2; *see also* claim [25] (§XII.B.13 above) in this Ground 2.

507. Moreover, a POSITA would have known that indexing is an old and common technique used well before 2004. Specifically, a POSITA would have understood that when some data changes on a disk or file system, it is inefficient to scan and re-index an entire system, only to reprocess unchanged files; instead, it is more efficient to index only files that have changed. This is easily achieved because storage and file systems include timestamps that denote when the files were last modified. Lastly, when some data changes, for example in any form of a simple database (such as an index), a POSITA would have understood that the new index

data would be easily merged with older data: this would be useful to purge now-stale index entries. *See* §VII.B above and §VII.F above.

508. A POSITA would have been motivated to combine Sapuntzakis and Federwisch for the reasons discussed in §XII.B.2 above.

509. Federwisch discloses a form of re-indexing of incremental snapshots (which the '844 Patent calls "leaf images"). In one example, "the Destination Snapshot A (2002) is now prepared to transfer changes so as to establish a mirror in Destination Snapshot B (2004). … In this case, both the new source and the new destination system have their own inode maps A and B (2012 and 2014), indexed off the old source 2001, and referencing the inodes in their respective trees. … It then treats the new source as the appropriate map index for the new associated inode map 2018. In the associated map, it stores the new source generation number for the new source index inode number, with each index entry also associated with/mapped to the new destination inode/generation number extracted from the inode map B (2014)." EX1008 (Federwisch), ¶[0137]. *See also* §X.C above.

510. *See* claim limitation [25] in this Ground 2 (§XII.B.13 above).

511. The combination of Sapuntzakis and Federwisch further discloses the instructions further causing the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-

indexing of said leaf image portion with said results of said indexing of said root image portion.

512. When snapshots in Federwisch are copied from a source to a destination, although the data has not changed, the snapshot's block locations may have changed. Thus, Federwisch updates the destination snapshot's index maps, a form of re-indexing.

513. In sum, Sapuntzakis and Federwisch further disclose the instructions further causing the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.

15. Claim 27: The computer-readable storage medium of claim 23 wherein said root image portion is read-only.

514. *See* claim limitations [23.0]-[23.4] (§XII.B.11 above) in this Ground 2, and claim [13] (§XII.A.8 above) in Ground 1 above.

C. Ground 3: Birse In View Of Holtzman Renders Claim 1-13 Obvious.

515. I discuss claims 7-13 first, followed by claims 1-6 (which recite similar limitations as claims 7-13).

1. Claim 7

a. [7.0] A method for providing data to a plurality of compute nodes, comprising:

516. Birse discloses a method for providing data (from NC server 170) to a plurality of compute nodes (NC clients 150, 812, 814, 816, 822, 824, 826): "…[T]he NC clients [150] boot over the network by accessing a stored copy of the operating system from the NC server [170]. Additionally, when NC clients want to run an application, they access it from the NC server." EX1006 (Birse), 4:56-60. *See also id.*, 2:51-3:14 (alternative embodiments providing data from NC server to NC clients to upgrade operating system, upgrade applications). This is illustrated in Birse's Figure 1, reproduced below:

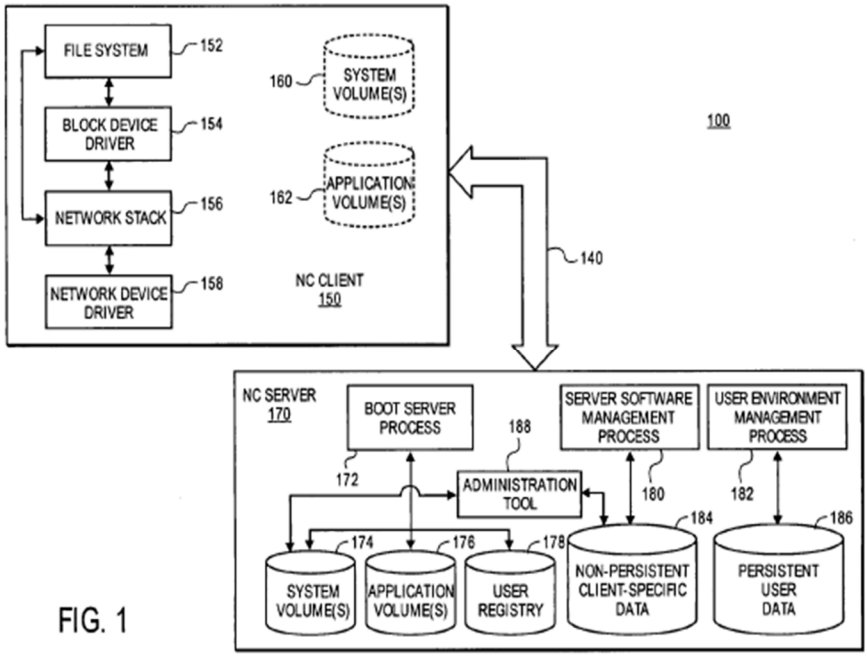
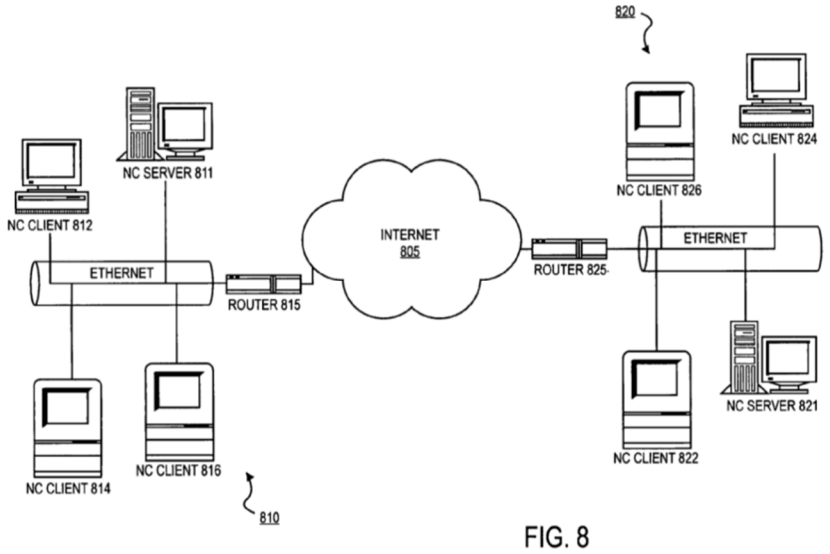


FIG. 1

Birse, Figure 1

517. "FIG. 8 is a simplified block diagram illustrating two NC systems 810 and 820 coupled via the Internet 805." *Id.*, 14:8-9. Birse's Figure 8 is reproduced below:



Birse, Figure 8

518. Further, NC server 170 and NC clients 150 are illustrated by exemplary digital processing system 200 of FIG. 2. As described at 7:63-8:35, they disclose all essential elements of a compute node. *Id.*, 4:27-57. Birse's Figure 2 is reproduced below:

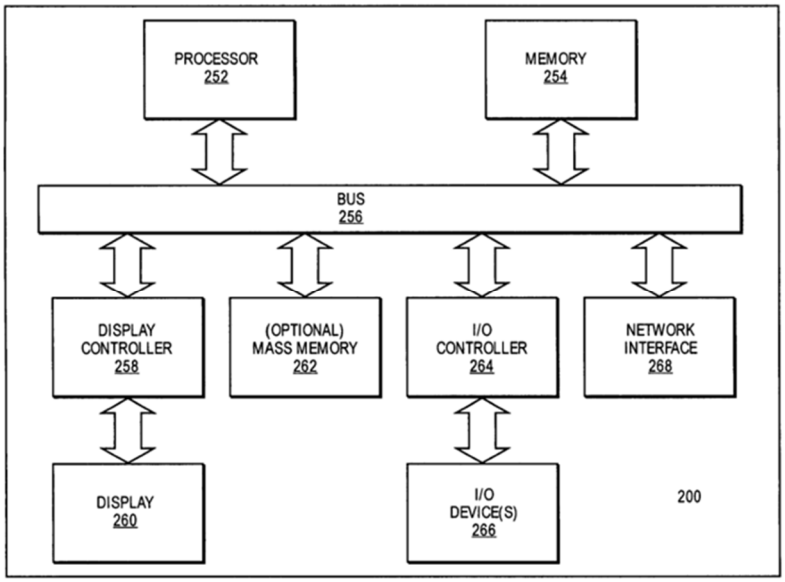


FIG. 2

Birse, Figure 2

519. In sum, Birse discloses a method for providing data (from NC server 170) to a plurality of compute nodes (NC clients 150, 812, 814, 816, 822, 824, 826).

b. [7.1] storing blocks of a root image of said compute nodes on a first storage unit;

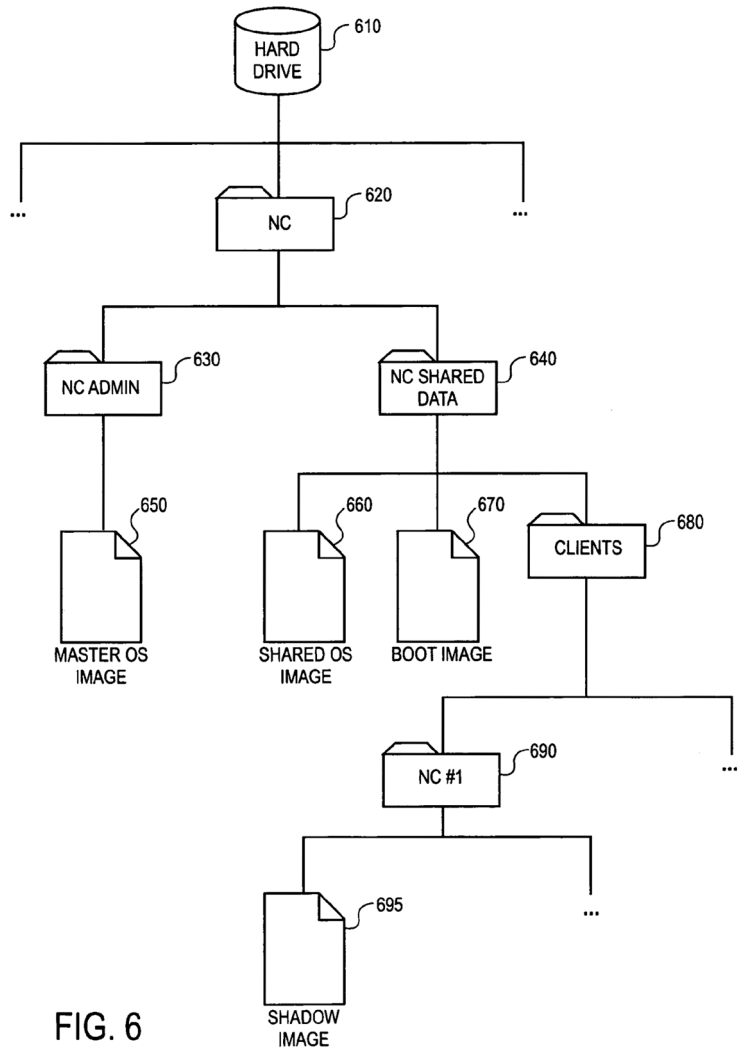
520. Birse discloses storing blocks of a root image (system volumes 174, 660, 922, 924) of said compute nodes (multiple NC servers 170) on a first storage unit (one or more hard drives 610, logically separate in NC folders/directories 630, 640): "The NC server 170 includes one or more system volumes 174…. The system volumes 174 include a protected, read-only, master copy of the operating system software." *Id*., 5:10-14.

521. "[T]he NC server 170 contains a read-only core system volume image 922 and a read-write user system volume image 924 … The core system volume 922

preferably contains those parts of the system that do not need to be written back to during system operation. One goal of the core system volume 922 is to provide all the system components that are mandatory for system operation…. The user system volume 924 contains all the user-configurable system components, including preferences and all the system additions installed by application software, such as application-installed extensions and libraries." *Id*., 14:27-34, 14:41-44.

522. "[T]he system volumes 174… may be independently distributed across multiple server computer systems." *Id*., 6:59-63.

523. "FIG. 6 is a simplified block diagram illustrating the layout of an exemplary hierarchical directory structure that may be used by an NC server [170] according to one embodiment…. [A] hard drive 610 of the NC server 170… includes an NC folder (directory) 620. Within the NC folder 620 are an NC admin folder 630 and an NC shared data folder 640." *Id*., 12:30-38. Birse's Figure 6 is reproduced below:



Birse, Figure 6

524. Birse discloses that file 660, storing the shared OS image (within the folder 640), is a first storage unit storing blocks of a root image. "The shared operating system image 660 is a read-write version of the read-only master operating system image 650 and may be mounted by the NC client 150 as part of step 395." *Id*., 12:44–47.

525. The shared system volume 660 (a.k.a. system volume 174, 922, 924, 1010) is stored as blocks as shown in Fig. 10, reproduced below:

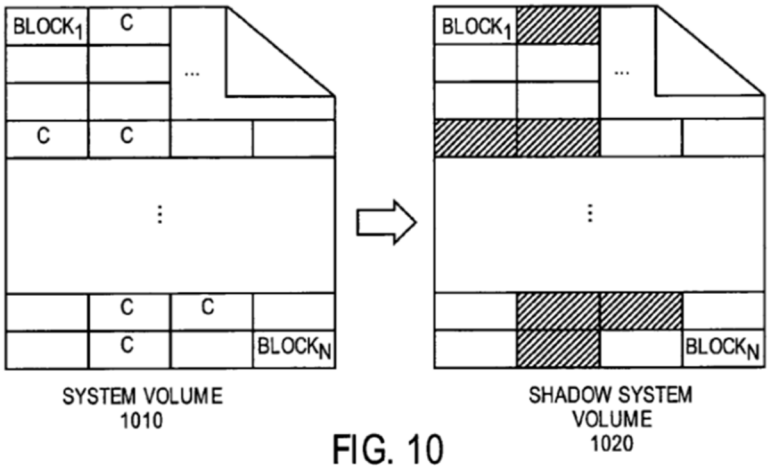


FIG. 10

Birse, Figure 10

526. In sum, Birse discloses storing blocks of a root image (system volumes 174, 660, 922, 924) of said compute nodes (multiple NC clients 150) on a first storage unit (one or more hard drives 610, logically separate in NC folders/directories 630, 640).

c. [7.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

527. Birse discloses storing leaf images (shadow volumes/images 930, 695) for respective compute nodes (NC clients 150, 812, 814, 816, 822, 824, 826) on respective second storage units (NC client folders 690 #1…n), said leaf images

including only additional data blocks not previously contained in said root image (write requests for new data) and changes made by respective compute nodes to the blocks of the root image (write request for changes to operating system data), wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes (only new data and changes are written): "In this example, the NC clients are numbered from 1 to N and the clients folder 680 includes a folder for each NC client. For example, an NC #1 folder 690 is an area for storing client-specific information, such as a shadow image 695, corresponding to NC client #1." *Id*., 12:54–59.

528. "The block device driver 154 services file system read and write requests to the system volumes 160. In one embodiment, the block device driver 154 appears to the file system 152 to be a standard hard drive device driver … [but is] configured to service the read and write requests of the file system 152 by accessing the NC server 170. As will be discussed further below, in one embodiment, one or more shadow volumes [695], may be created for the NC client 150 and stored with the non-persistent client-specific data 184 for the purpose of storing user preferences and/or changes to the operating system. Therefore, some mechanism is needed to direct reads and writes to the appropriate volume, e.g., the shadow volumes or the system volumes 174. According to one embodiment, the block device driver 154 has knowledge of the separate shadow volumes and system

volumes 174 and redirects read and write requests to the appropriate volume. Alternatively, server-side logic may perform the redirection locally." *Id.*, 7:35-53; *see also id.*, 11:30-32.

529. "FIG. 4 is a flow diagram illustrating how NC client [150] write requests directed to the system volumes 160 are handled… [A]t step 410, the file system 152 generates a write request directed to the system volumes 160. The write request is received by the block device driver 154 and redirected by the block device driver 154 at step 420 by translating it into a write request directed to the user's shadow volume on the NC server 170. At step 430, the NC server 170 stores information associated with the write request in the shadow volume associated with the NC client 150. In this manner, the file system 152 need not be aware that attempted modifications to the system volumes 160 are recorded instead in a client-specific shadow volume residing on the NC server 170." *Id.*, 10:63-65, 11:6-17. Birse's Figure 4 is reproduced below:

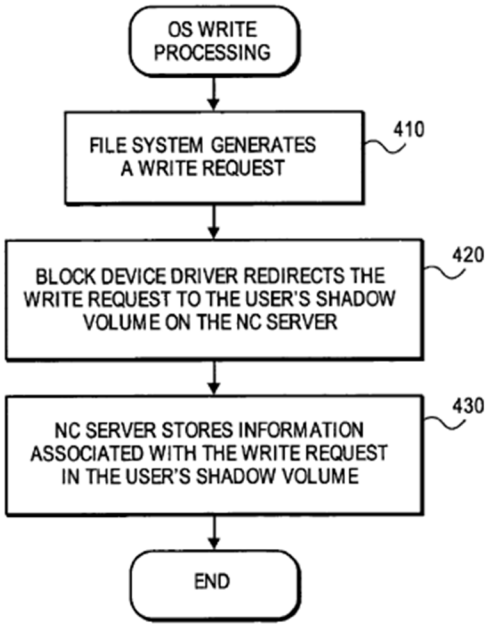


FIG. 4

Birse, Figure 4

530. In one embodiment, "the shadow volume contains only those portions of the operating system that have been modified by the user rather than a complete copy of the operating system as modified." *Id*., 11:49-52.

531. In another embodiment, the shared OS image 660 is divided into two parts: "[t]he NC server 170 also creates a shadow system volume 930 for each connected NC client. The shadow system volume 930 shadows the user system volume 924 by storing modifications that are made to the user system volume 924." *Id*., 14:49-53.

532. Moreover, "… the shadow system volume 930 is used by the block device driver 154 of the NC client 150 to implement a 'copy-on-write' storage scheme in which modifications (writes) directed to the user system volume 930 are instead copied to the shadow system volume 930." *Id*., 14:56-61.

533. In sum, Birse discloses storing leaf images (shadow volumes/images 930, 695) for respective compute nodes (NC clients 150, 812, 814, 816, 822, 824, 826) on respective second storage units (NC client folders 690 #1…n), said leaf images including only additional data blocks not previously contained in said root image (write requests for new data) and changes made by respective compute nodes to the blocks of the root image (write request for changes to operating system data), wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes (only new data and changes are written).

d. [7.3] caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.

534. Birse and Holzmann disclose caching blocks of said root image (system volumes 174, 660, 922, 924) that have been accessed by at least one of said compute nodes (NC clients 150, 812, 814, 816, 822, 824, 826) in a cache memory. EX1006 (Birse), 4:56-60, 5:9-29, 6:43-48; EX1007 (Holzmann), Abstract, [0021], [0024].

535. See Holzmann portion of [7.3] of Ground 1, above (§XII.A.1.d above).

536. It would have been obvious to a POSITA to combine the cache system of Holzmann with the system of Birse to improve the performance of Birse, as described next. *See also* §X.B above.

2. Motivation to Combine Birse and Holzmann

537. A POSITA would have been motivated to combine Birse and Holzmann for at least the following five reasons.

538. **First**, Birse and Holzmann are both concerned with the management of whole operating system (OS) images and booting systems over the network.

539. In Birse, "[t]he NC shared data folder 640 includes a shared operating system image 660, a boot image 670, and a clients folder 680. The shared operating system image 660 is a read-write version of the read-only master operating system image 650 and may be mounted by the NC client 150 as part of step 395." EX1006 (Birse), 12:42-47. Birse is further concerned with "providing a reliable and maintainable operating system in a net-booted environment." EX1006 (Birse), 1:55-57.

540. Similarly, Holzmann "is directed to a system and method for booting multiple servers or other network resources from a single operating system image." EX1007 (Holzmann), Abstract.

541. **Second**, Birse and Holzmann are both concerned with the efficient management of these OS images.

542. Birse discloses that "[t]he main motivation for banding is to reduce disk space requirements of the NC client shadow system images. According to one embodiment, the reduction is achieved by only storing those bands which have actually been written to." EX1006 (Birse), 15:41-45.

543. Holzmann similarly recognizes some of the core elements that improve system efficiency: faster storage devices and shared network storage systems. "SSD systems can access data more quickly and process more read and write requests than conventional disk systems found on most server computers." EX1007 (Holzmann), ¶[0004]. "Booting servers from a SAN offers numerous advantages. The SAN allows the various operating system images to be stored centrally, allowing for efficient loading, monitoring, patching and updating of the operating system. Central storage of operating system images on the SAN also facilitates easier replacement or swapping of server hardware." EX1007 (Holzmann), ¶[0008].

544. **Third**, both Birse and Holzmann disclose caching blocks of the root/boot disk images.

545. Birse mentions common caching features that any operating system would use. *See* §VII.D above. "The processor may also be coupled to other types of storage areas/memories (e.g., cache, Flash memory, disk, etc.), which could be considered as part of the memory 254 or separate from the memory 254." EX1006 (Birse), 8:17-20. Moreover, "Macintosh and Intel systems often have multiple

busses, such as a peripheral bus, a dedicated cache bus, etc." EX1006 (Birse), 8:39-41.

546. While Birse discloses commonly understood caching techniques, it does not detail how caching may be applied to its boot images. A POSITA would still have understood that Birse's clients and servers are regular computers that use typical caching.

547. Holzmann, however, discloses details of caches for each system/server being booted, and different cache configurations.

548. Holzmann suggests various forms of caches. "Alternatively, the present system may be configured to allow for dynamic allocation of cache resources through the use of a non-associative cache, as depicted in FIG. 5." EX1007 (Holzmann), ¶[0034]. "[S]till more cache configurations can be used in accordance with the invented system, including additional set-associative caches or fully associative caches." EX1007 (Holzmann), ¶[0038].

549. Holzmann discloses specific examples where each server or operating system has its own dedicated cache.

550. "As multiple servers or other external devices 106 are booted, the control module 103 allocates additional space into the cache stored in memory module 104. Portions of the cache may be assigned statically for each server or other external devices 106. Alternatively, and preferably, the portions of the cache

attributed to each server may be assigned dynamically, as described further below with reference to FIG. 5." EX1007 (Holzmann), ¶[0021], Fig. 5 (reproduced below).

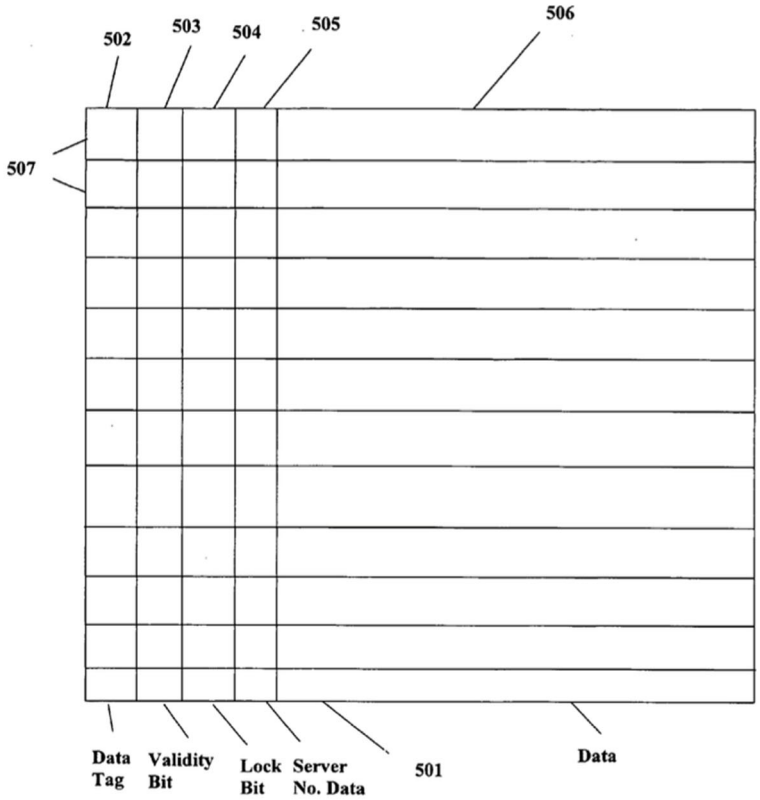


Fig. 5

Holzmann, Figure 5

551. Furthermore, "[b]y allocating separate cache space for each server or other network resource, the inventive system prevents corruption of data, overwriting of data, and other harmful actions that would lead to resource malfunction." EX1007 (Holzmann), ¶[0033].

552. **Fourth**, a POSITA would have been motivated to modify Birse with Holzmann's caching techniques. Birse discloses typical OS caching but a POSITA

would have known that there are many different caching techniques with their own pros and cons, as detailed in §VII.D above.

553. Specifically, and as discussed immediately above, Holzmann offers details on how to cache data for each operating system (*e.g.*, Birse's boot images). Holzmann goes further and discloses several forms of caches: unique caches per server, caches that are statically or dynamically allocated, non-associative caches, set-associative caches, and fully associative ones. EX1007 (Holzmann), ¶¶[0021], [0029], [0034], [0037], [0038].

554. **Fifth** and lastly, a POSITA would have known that caching is an age-old technique that has been commonly used in various locations for decades before the '844 Patent. *See* §VII.D above. In addition, caching data in RAM memory is as old as computers themselves. *See* §VII.A above. A POSITA interested in even more details of caching than either Birse or Holzmann disclose could easily consult the extensive literature on the subject.

555. Thus, modifying Birse with Holzmann's caches would have been simple and would not have required extensive experimentation or impose undue burden, because both already disclose caching. For the same reasons, the combination would have yielded predictable results (because caching is well understood to improve performance and resource utilization). Lastly, a POSITA

would have had a high expectation of success because Holzmann does nothing more than describe caching in more detail than Birse does.

3. Claim 8

a. [8.1] The method as recited in claim 7 further comprising: receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and

556. *See* claim limitations [7.0]-[7.3] in this Ground 3 (§515 above).

557. Birse and Holzmann disclose receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory.

558. *See* Holzmann portion of claim [8.1] in Ground 1 above (§XII.A.3.a above).

559. It would have been obvious to a POSITA to combine the cache system of Holzmann with the system of Birse to improve the performance of Birse. *See* XII.C.2 above.

b. [8.2] providing said first portion of said data to said at least one of said compute nodes from said cache memory.

560. Birse and Holzmann disclose providing said first portion of said data to said at least one of said compute nodes from said cache memory.

561. *See* Holzmann portion of claim [8.2] in Ground 1 above (§XII.A.3.b above).

562. It would have been obvious to a POSITA to combine the cache system of Holzmann with the system of Birse to improve the performance of Birse. *See* §XII.C.2 above.

4. Claim 9: The method as recited in claim 8 further comprising: updating said cache memory based on said read request.

563. *See* claim limitations [8.1]-[8.2] in this Ground 3 (§XII.C.3 above). *See* Holzmann portion of claim [9] in Ground 1 above (§XII.A.4 above).

5. Claim 10

a. [10.0] The method as recited in claim 9

564. *See* claim [9] in this Ground 3 (§XII.C.4 above).

b. [10.1] wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises: caching said second portion in said cache memory; and

565. *See* Holzmann portion of claim [10.1] in Ground 1 above (§XII.A.5.b above).

c. [10.2] removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value.

566. *See* Holzmann portion of claim [10.2] in Ground 1 above (§XII.A.5.c above515 above).

6. Claim 11: The method as recited in claim 7 further comprising: merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments.

567. *See* claim limitations [7.0]-[7.3] in this Ground 3 (§515 above).

568. Birse discloses merging the blocks of said root image (blocks of the operating system software stored on system volumes 174, 660, 922) with the blocks of respective leaf images (695) to create cohesive respective application environments (dynamically merged system environment): "In one embodiment, the shadow system volume 930 is not visible to the user. Consequently, the user system volume 924 will appear to be the location where all writes are directed. For example, if a user modifies a preference, that preference will appear to be written to the preference file located in the preferences folder on the user system volume 924." EX1006 (Birse), 14:61-67.

569. "When an NC client [150] boots from the network and accesses the operating system from an NC server [170], the user's preferences are dynamically merged with the system environment provided to the NC client [150]. Advantageously, since the user's desktop preferences and other customized settings are all preserved from session to session and supplied to the NC client [150] as it boots from the network, the user may login to any NC client [150] on the network and have the same user experience." *Id*., 9:2-10.

570. "Referring now to Fig. 5, NC client [150] read request processing will now be described according to one embodiment of the present invention. … At step 510, the file system 152 generates a read request directed to the system volumes 160. At step 520, the read request is received by the block device driver 154 and a determination is made whether the read request specifies a portion of the operating system that has been modified by the user or whether the read request specifies a portion of the operating system that remains unchanged. If the read request corresponds to a portion of the operating system that has not been modified by the user, the processing continues with step 530. However, if the read request corresponds to a portion of the operating system that has been previously modified by the user, then processing continues with step 540. At step 530, the block device driver directs the NC server 170 to retrieve the information associated with the read request from the system volumes 174. At step 540, the block device driver directs the NC server 170 to retrieve the information associated with the read request from the user's shadow volume. In this manner, the file system 152 need not be aware of the particular mechanism used to coordinate user modifications with the original operating system as contained in the system volumes 174." *Id.*, 11:39-41, 11:55-12:10. Birse's Figure 5 is reproduced below:

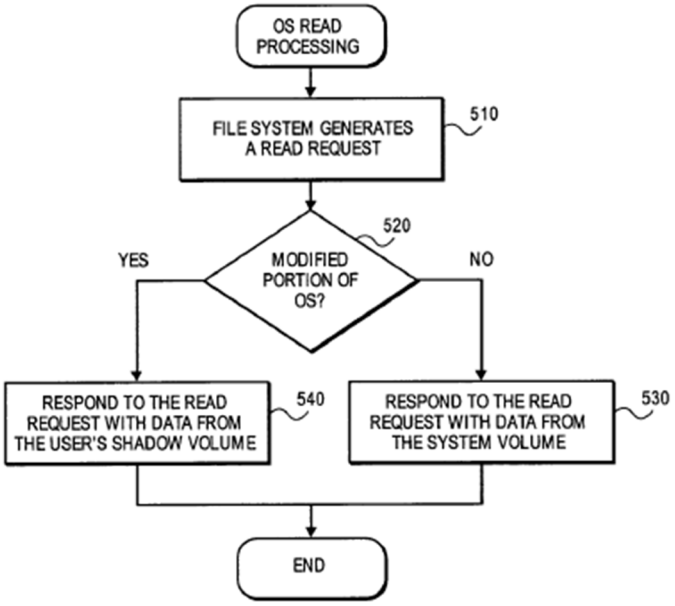


FIG. 5

Birse, Figure 5

571. In sum, Birse discloses merging the blocks of said root image (blocks of the operating system software stored on system volumes 174, 660, 922) with the blocks of respective leaf images (695) to create cohesive respective application environments (dynamically merged system environment).

7. Claim 12: The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory.

572. The combination of Birse and Holzmann discloses the method as recited in claim 11, and, in particular, that the block device drivers (which serve as

the union block devices) work below the client file system, and, hence, at an operational level. *See* claim limitation [11] in this Ground 3 (§XII.C.6 above).

573. Because the block device drivers (154) of Birse are already designed to merge blocks being read from multiple sources into one cohesive image using the algorithm of Figure 5, it would have been obvious to a POSITA to combine Birse with Holzmann to extend the block device drivers' function to incorporate blocks stored in the cache, as well. EX1006 (Birse), 11:39-12:26. That is because cached blocks are just as easily accessible to an operating system or storage system as blocks stored on any storage device such as a hard disk. *See also* §VII.A above and §VII.D above.

574. A POSITA would have been motivated to combine Birse with Holzmann for the reasons disclosed in § XII.C.2 above.

575. In sum, Birse in view of Holzmann discloses the method as recited in claim 11, and, in particular, that the block device drivers 154 of Birse (which serve as the union block devices) work below the client file system, and, hence, at an operational level.

8. Claim 13: The method as recited in claim 7 wherein said first storage unit is read-only.

576. *See* claim limitations [7.0]-[7.3] in this Ground 3 (§515 above).

577. Birse further discloses the first storage unit being read-only. "The system volumes **174** include a protected, read-only, master copy of the operating system software." EX1006 (Birse), 5:13-14.

578. In sum, Birse discloses the first storage unit being read-only.

9. Claim 1

a. [1.0] A system for providing data to a plurality of compute nodes, the system comprising:

579. *See* claim limitation [7.0] in this Ground 3 (§XII.C.1.a above).

b. [1.1] a first storage unit configured to store blocks of a root image of said compute nodes;

580. *See* claim limitations [7.1] in this Ground 3 (§XII.C.1.b above).

c. [1.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

581. *See* claim limitations [7.2] in this Ground 3 (§XII.C.1.c above).

d. [1.3] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes.

582. *See* claim limitations [7.3] in this Ground 3 (§XII.C.1.d above).

10. Claim 2: The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value.

583. *See* claim limitations [1.0]-[1.3] in this Ground 3 (§XII.C.9 above).

584. Moreover, this limitation describes nothing more than common caching behavior (freeing some items from a cache whose size is by definition limited, to make room for new items). *See* §VII.D above.

585. Holzmann states that "[i]f the requested data has not been loaded into the cache 501, the control module 103 copies the data from the operating system data stored in memory portion 400 to the cache 501." EX1007 (Holzmann), ¶[0035].

586. Moreover, "[i]t will be appreciated by those skilled in the art that other cache configurations may be used in accordance with the present invention, such as a set associative cache." EX1007 (Holzmann), ¶[0037]. "It will be understood by those skilled in the art that still more cache configurations can be used in accordance with the invented system, including additional set-associative caches or fully associative caches." EX1007 (Holzmann), ¶[0038].

587. A POSITA would have understood this limitation describes nothing more than common caching behavior (cache management policies, where caches naturally have limited room and must use thresholds to decide what to keep/evict). *See* §VII.D above (especially LRU and LFU cache-management policies).

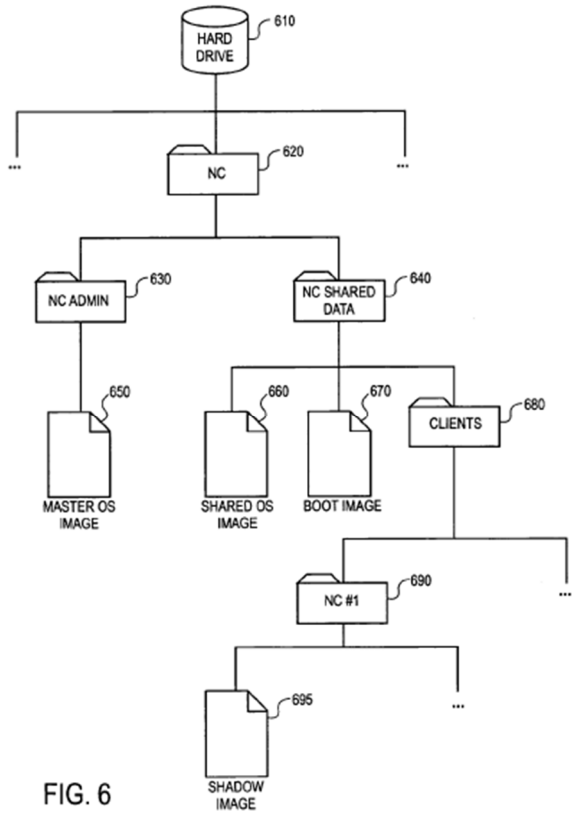
588. In sum, Holzmann discloses said cache being configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value.

11. Claim 3: The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance.

589. *See* claim limitations [1.0]-[1.3] in this Ground 3 (§XII.C.9 above).

590. Birse discloses the first storage unit (holding the core system volumes and the user system volume), and the second storage units (holding the shadow system volumes) contained within a single storage appliance: "FIG. 6 is a simplified block diagram illustrating the layout of an exemplary hierarchical directory structure that may be used by an NC server according to one embodiment of the present invention." EX1006 (Birse), 12:30–33.

591. "According to the embodiment depicted, a hard drive 610 of the NC server 170 or other mass storage device associated with the NC server 170 includes an NC folder (directory) 620." EX1006 (Birse), 12:34–37. Birse's Figure 6 is reproduced below:



Birse, Figure 6

592. Holzmann discloses a cache mechanism for server drives. "The control module 103 then allocates a sub-portion of the memory portion to be used as a cache for the operating system. As multiple servers or other external devices 106 are booted, the control module 103 allocates additional space into the cache stored in memory module 104. Portions of the cache may be assigned statically for each server or other external devices 106. Alternatively, and preferably, the portions of the cache attributed to each server may be assigned dynamically, as described further below with reference to FIG. 5." EX1007 (Holzmann), ¶[0021].

593. It would have been obvious to a POSITA to implement the cache system of Holzmann on the storage device holding the first and second storage units. This would also improve the cost/benefit of the use of high-speed solid state, persistent disk-centered memory caches (of the sort present in certain embodiments of Holzmann). For example, a POSITA would have known that the storage systems created by Netapp are all single storage appliances; each such appliance supports multiple volumes, multiple base images, multiple snapshots, and multiple caches (both volatile and non-volatile), as I discuss in §VII.E above.

594. In sum, Holzmann discloses said first storage unit, said second storage units, and said cache being contained within a single storage appliance.

12. Claim 4: The system as recited in claim 1 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.

595. *See* claim limitations [1.0]-[1.3] (§XII.C.9 above) and [12] (§XII.C.7 above) in this Ground 3.

596. A POSITA would have understood that such system components, taken together, embody the Union Block Device of this claim. For example, a POSITA would have known that the storage systems created by Netapp are all single storage appliances; each such appliance supports multiple volumes, multiple base images,

multiple snapshots, and multiple caches (both volatile and non-volatile). A POSITA would have understood each volume in a Netapp or similar appliance to be the claimed UBD. *See* §VII.E above.

597. In sum, Birse discloses a plurality of union block devices configured to interface between respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image (the system volumes) with the blocks of respective leaf images (the shadow volumes).

13. Claim 5: The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.

598. *See* claims [4] (§XII.C.12 above) and [12] (§XII.C.7 above) in this Ground 3.

14. Claim 6: The system as recited in claim 1 wherein said first storage unit is read-only.

599. *See* claim limitations [1.0]-[1.3] (§XII.C.9 above) and [13] (§XII.C.8 above) in this Ground 3.

D. Ground 4: Birse In View Of Federwisch Renders Claim 14-27 Obvious.

1. Claim 14

a. [14.0] A system for indexing file systems for a plurality of compute nodes, the system comprising:

600. *See* claim limitation [7.0] in Ground 3 above (§XII.C.1.a above).

b. [14.1] a first storage unit configured to store blocks of a root image of said compute nodes;

601. *See* claim limitation [7.1] in Ground 3 above (§XII.C.1.b above).

c. [14.2] a plurality of second storage units configured to store leaf images of respective compute nodes, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

602. *See* claim limitation [7.2] in Ground 3 above (§XII.C.1.c above).

d. [14.3]-[14.4] a plurality of union block devices corresponding to said compute nodes, said union block devices configured to interface between said compute nodes and said first and second storage units to distribute said file systems to said compute nodes, wherein said union block devices are configured to create said file systems by merging the blocks of said root image stored on the first storage unit with the blocks of respective leaf images stored on respective second storage units, and wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.

603. Birse and Federwisch further disclose at least one of said compute nodes being configured to index said root image and provide the indexing results to another of said compute nodes.

604. *See* Federwisch portion of Ground 2 in claim limitation [14.3]-[14.4] (§XII.B.1.d above).

605. Birse discloses the remaining limitations of this term. *See* claim limitation [4] in Ground 3 above (§XII.C.12 above) and claim limitation [11] in Ground 3 above (§XII.C.12 above).

606. As an initial matter, a POSITA would have known that indexing is an old and common technique used well before 2004. *See* §VII.F above

607. Federwisch discloses a "system and method for remote asynchronous replication or mirroring of changes" between "a source file system snapshot" and "a destination replica file system." EX1008 (Federwisch), Abstract.

608. Federwisch compares two snapshots "using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file system, which identifies changed blocks in the respective snapshot files based upon differences in volume block numbers identified in a scan of the logical file block index of each snapshot." EX1008 (Federwisch), Abstract. Thus, every snapshot in a volume has a "logical file block index."

609. A POSITA would have been motivated to combine Birse and Federwisch for the reasons discussed in §XII.D.2 below.

610. Birse suggests that the boot images are indexed and organized through boot file selection and a registry. "In order to allow the NC server 320 to recognize the NC client 310 next time it boots, information regarding the NC client 310 may be stored in the user registry 178." EX1006 (Birse), 10:5-8. Then, "[i]f the NC client 310 is known to the NC server 320, then at step 365 information regarding the NC client 310, such as its IP address may be retrieved from the user registry 178." EX1006 (Birse), 10:16-18.

611. Because Birse only suggests the use of indexing, a POSITA interested in building a system based on Birse would have been motivated to seek out additional details on indexing storage systems, files, and caches thereof. This is particularly so because indexing of data items was a well-known technique. *See*

§VII.F above. Such a POSITA would have found Federwisch, which discloses greater details of indexing. *See* §X.C above.

612. Federwisch compares two snapshots "using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file system, which identifies changed blocks in the respective snapshot files based upon differences in volume block numbers identified in a scan of the logical file block index of each snapshot." EX1008 (Federwisch), Abstract. Thus, every snapshot in Federwisch's volumes has a "logical file block index."

613. Federwisch uses common indexing to better manage cached data. "If the information is not in memory, the file system layer 450 indexes into the inode file using the inode number to access an appropriate entry and retrieve a volume block number." EX1008 (Federwisch), ¶[0053].

614. Federwisch consults its indexes to identify changed and unchanged blocks. "The existence of a changed volume block number at a given offset in the index between the base snapshot inode file and incremental snapshot inode file generally indicates that one or more of the underlying inodes and files to which the inodes point have been changed." EX1008 (Federwisch), ¶[0071].

615. Federwisch scans these indexes to identify unchanged blocks. "A scanner 820 searches the index for changed base/incremental inode file snapshot blocks, comparing volume block numbers or another identifier. … Hence, scanned

blocks at the same index in both snapshots can be efficiently bypassed, reducing the scan time." EX1008 (Federwisch), ¶[0072].

616. In sum, Birse and Federwisch disclose at least one of said compute nodes being configured to index said root image and provide the indexing results to another of said compute nodes.

2. Motivation to Combine Birse with Federwisch

617. A POSITA would have been motivated to combine Birse and Federwisch for at least the following six reasons.

618. **First**, both operate at and recognize disk blocks at the block layer. A POSITA having reviewed Birse would have naturally found other references in the same field of operating on block layer storage systems. EX1006 (Birse), 15:24-28; EX1008 (Federwisch), Abstract.

619. In Birse, "the sparse shadow system volume 1120 is written in band increments rather than as an entire file. A band is a predetermined number of blocks. For example, according to one embodiment bands are 128K (256 512-byte blocks)." EX1006 (Birse), 15:24-28. Moreover, "the block device driver 154 may redirect read and write requests received from the file system 152 to a remote storage medium on the NC server, for example." EX1006 (Birse), 10:59-62.

620. Similarly, Federwisch discloses "[a] system and method for remote asynchronous replication or mirroring of changes in a source file system snapshot in

a destination replica file system using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file system, which identifies changed blocks in the respective snapshot files based upon differences in volume block numbers identified in a scan of the logical file block index of each snapshot." EX1008 (Federwisch), Abstract.

621. **Second**, both Birse and Federwisch disclose and use CoW snapshots. A POSITA would have known that CoW techniques enable snapshots. *See* §VII.E above.

622. I note that while Birse uses snapshots, it calls them "shadow volumes." "In the embodiment depicted, it is assumed that the shadow volume contains only those portions of the operating system that have been modified by the user rather than a complete copy of the operating system as modified. The various options regarding the granularity of the portions written to the shadow volume are discussed below." EX1006 (Birse), 11:49-54. A POSITA would have understood Birse to disclose incremental snapshots that contain only changed blocks relative to a base, read-only image. *See also* §VII.E above, EX1006 (Birse), Fig. 9, reproduce below.

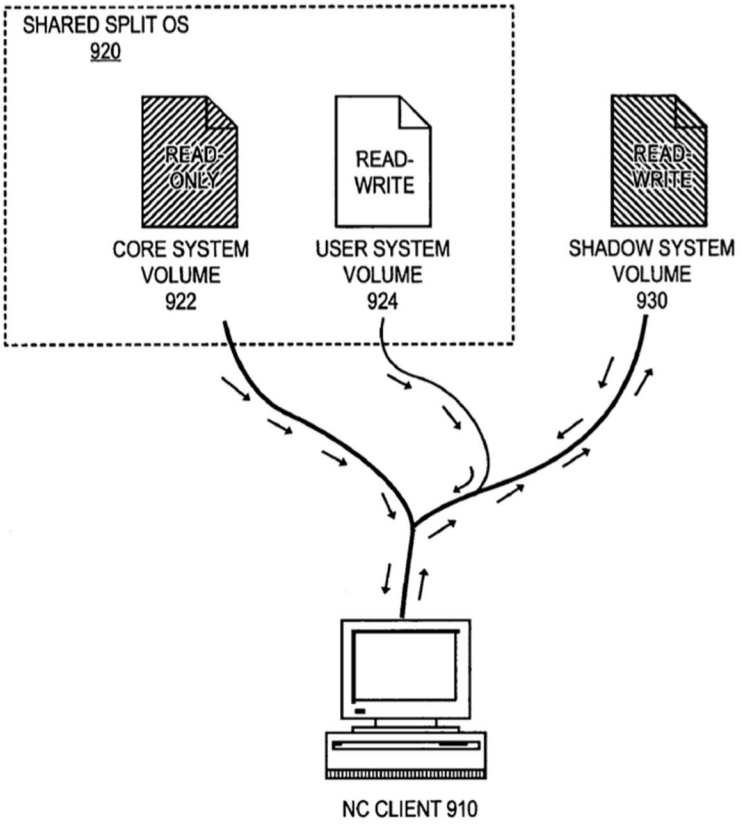


FIG. 9

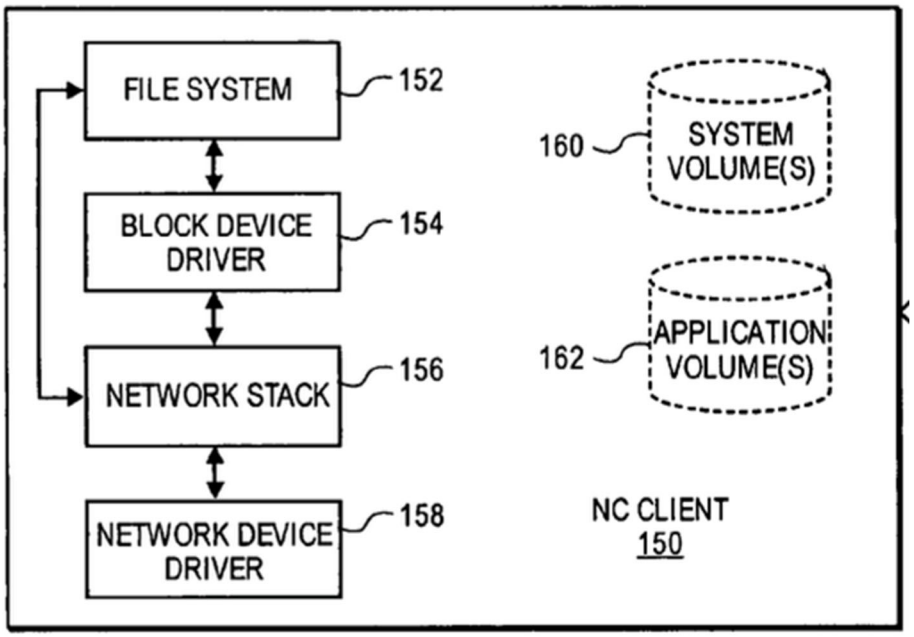
Birse, Figure 9

623. *Compare* Birse, 15:33-37 ("According to one embodiment of the banding feature, the inherent one-to-one block mapping in copy-on-write images may be replaced with a table or other data structure that maps a band of the primary file to a corresponding band in the shadow file.") *with* Federwisch, Abstract ("A system and method for remote asynchronous replication or mirroring of changes in a source file system snapshot in a destination replica file system using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file

system, which identifies changed blocks in the respective snapshot files based upon differences in volume block numbers identified in a scan of the logical file block index of each snapshot.”).

624. **Third**, while both operate at the block layer, they recognize that file systems interoperate and work atop the block layer.

625. In Birse, “[t]he block device driver 154 services file system read and write requests to the system volumes 160. In one embodiment, the block device driver 154 appears to the file system 152 to be a standard hard drive device driver. However, in reality, the block device driver 154 may be configured to service the read and write requests of the file system 152 by accessing the NC server 170.” EX1006 (Birse), 7:35-41; *see also id*., Fig. 1 (excerpt reproduced below).



Birse, Figure 1 (excerpted)

626. Similarly, Federwisch discloses "[a] system and method for remote asynchronous replication or mirroring of changes in a source file system snapshot in a destination replica file system using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file system, which identifies changed blocks in the respective snapshot files based upon differences in volume block numbers identified in a scan of the logical file block index of each snapshot." EX1008 (Federwisch), Abstract.

627. **Fourth**, as discussed in §XII.D.2 above, Birse discloses caching blocks of the root/boot disk images. Being such a common technique, it is no surprise that Federwisch also discloses at least two forms of caches: volatile and non-volatile. *See also* §VII.A above.

628. Federwisch caches data in volatile RAM memory caches. "The snapshot is stored on-disk along with the active file system, and is called into the buffer cache of the filer memory as requested by the storage operating system 400 or snapshot mirror application 490 as described further below." EX1008 (Federwisch), ¶[0058]. Moreover, "[t]he inode files may each be loaded into the buffer cache of the source file server memory from the on-disk versions thereof using conventional disk access processes as directed by the storage operating system snapshot manager (480 in Fig. 4)." EX1008 (Federwisch), ¶[0070].

629. Federwisch also caches data in non-volatile, batter-backed RAM (NVRAM). "The NVRAM is filled, in parallel with the buffer cache, after each client request is completed, but before the result of the request is returned to the requesting client." EX1008 (Federwisch), ¶[0048]. *See also* §VII.E above.

630. **Fifth**, a POSITA would have been motivated to modify Birse with Federwisch's indexing.

631. Birse suggests that the boot images are indexed and organized through boot file selection and a registry. "In order to allow the NC server 320 to recognize the NC client 310 next time it boots, information regarding the NC client 310 may be stored in the user registry 178." EX1006 (Birse), 10:5-8. Then, "[i]f the NC client 310 is known to the NC server 320, then at step 365 information regarding the NC client 310, such as its IP address may be retrieved from the user registry 178." EX1006 (Birse), 10:16-18.

632. Because Birse only suggests the use of indexing, a POSITA interested in building a system based on Birse would have been motivated to seek out additional details on indexing storage systems, files, and caches thereof. This is particularly so because indexing of data items was a well-known technique. *See* §VII.F above. Such a POSITA would have found Federwisch, which discloses greater details of indexing. *See* §X.C above.

633. Federwisch compares two snapshots "using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file system, which identifies changed blocks in the respective snapshot files based upon differences in volume block numbers identified in a scan of the logical file block index of each snapshot." EX1008 (Federwisch), Abstract. Thus, every snapshot in Federwisch's volumes has a "logical file block index."

634. Federwisch uses common indexing to better manage cached data. "If the information is not in memory, the file system layer 450 indexes into the inode file using the inode number to access an appropriate entry and retrieve a volume block number." EX1008 (Federwisch), ¶[0053].

635. Federwisch consults its indexes to identify changed and unchanged blocks. "The existence of a changed volume block number at a given offset in the index between the base snapshot inode file and incremental snapshot inode file generally indicates that one or more of the underlying inodes and files to which the inodes point have been changed." EX1008 (Federwisch), ¶[0071].

636. Federwisch scans these indexes to identify unchanged blocks. "A scanner 820 searches the index for changed base/incremental inode file snapshot blocks, comparing volume block numbers or another identifier. … Hence, scanned blocks at the same index in both snapshots can be efficiently bypassed, reducing the scan time." EX1008 (Federwisch), ¶[0072].

637. Federwisch even discloses a form of re-indexing of incremental snapshots (which the '844 Patent calls "leaf images"). In one example, "the Destination Snapshot A (2002) is now prepared to transfer changes so as to establish a mirror in Destination Snapshot B (2004). … In this case, both the new source and the new destination system have their own inode maps A and B (2012 and 2014), indexed off the old source 2001, and referencing the inodes in their respective trees. … It then treats the new source as the appropriate map index for the new associated inode map 2018. In the associated map, it stores the new source generation number for the new source index inode number, with each index entry also associated with/mapped to the new destination inode/generation number extracted from the inode map B (2014)." EX1008 (Federwisch), ¶[0137]. *See also* §X.C above.

638. **Sixth** and lastly, a POSITA would have known that indexing of any data is an age-old technique long utilized in computer systems. *See* §VII.F above. Just as a library catalog provides an efficient way of locating books on shelves, so do computers create, manage, and update indexes of many data types (files, disk blocks, images, virtual machines, users, networks, and many more).

639. Thus, modifying Birse to include Federwisch's indexing would have been simple and would not have required extensive experimentation or impose undue burden, because—most likely—Birse already indexed all of their data objects.

640. For the same reasons, the combination of Birse and Federwisch would have yielded predictable results (because indexes are well understood to structure and locate data objects efficiently). Lastly, a POSITA would have had a high expectation of success because indexing techniques have been well understood for decades prior to 2004.

3. Claim 15: The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance.

641. *See* claim limitations [14.0]-[14.4] (§XII.D.1 above) in this ground 4, and claim limitation [3] (§XII.C.11 above) in Ground 3 above.

4. Claim 16: The system as recited in claim 14 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit and respective second storage units, said union block devices configured to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.

642. *See* claim limitations [14.0]-[14.4] (§XII.D.1 above) in this ground 4, claim limitation [4] (§XII.C.12 above) in Ground 3 above.

5. Claim 17: The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.

643. *See* claim limitation [16] (§XII.D.4 above) in this ground 4, and claim limitation [5] (§XII.C.13 above) in Ground 3 above.

6. Claim 18: The system as recited in claim 14 wherein said first storage unit is read-only.

644. *See* claim limitations [14.0]-[14.4] (§XII.D.1 above) in this ground 4, and claim limitation [13] (§XII.C.8 above) in Ground 3 above.

7. Claim 19

a. [19.0] A method for indexing file systems for a plurality of compute nodes, comprising:

645. *See* claim limitation [14.0] in this ground 4 (§XII.D.1.a above).

b. [19.1] storing blocks of a root image of said compute nodes on a first storage unit;

646. *See* claim limitation [14.1] in this ground 4 (§XII.D.1.b above).

c. [19.2] storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images for respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes;

647. *See* claim limitation [14.2] in this ground 4 (§XII.D.1.c above).

d. [19.3] merging the blocks of said root image with the blocks of respective leaf images stored on respective second storage units to create respective file systems for respective compute nodes;

648. *See* claim limitation [11] in ground 3 (§XII.C.6 above).

e. [19.4] receiving indexing results pertaining to said root image from one of said compute nodes; and

649. *See* claim limitation [14.4] in this ground 4 (§XII.D.1.d above).

f. [19.4] providing said indexing results to the others of said compute nodes.

650. *See* claim limitation [14.4] in this ground 4 (§XII.D.1.d above).

8. Claim 20: The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit.

651. *See* claim limitation [14.4] in this Ground 4 (§XII.D.1.d above).

652. As an initial matter, a POSITA would have known that sharing storage and files across devices, say over the network, was well known well before 2004. This would have made any data (index or otherwise) conveniently accessible over network storage volumes. *See* §VII.C above.

653. A POSITA would have been motivated to combine Birse and Federwisch for the reasons discussed in §XII.D.2 above.

654. *See* claim limitations [19.0]-[19.4] in this Ground 4 (§XII.D.7 above).

655. Birse and Federwisch further disclose storing said indexing results on a shared storage unit.

656. Federwisch recognizes the usefulness of sharing files and data in a format-agnostic manner to other systems. "More generally, the format should also be independent of the underlying network protocol or device (in the case of a tape or local disk/non-volatile storage) protocol and file system—that is, the information is system 'agnostic,' and not bound to a particular operating system software, thereby allowing source and destination systems of different vendors to share the information. The format should, thus, be completely self-describing requiring no information outside the data stream." EX1008 (Federwisch), ¶[0086].

657. Being a network appliance, Federwisch supports the ability to share files, snapshots, and whole volumes across the network (*e.g.*, using NFS or CIFS). This sharing is called "an export." "The destination file system is exported read-only the user." *Id.*, ¶[0015]. This first snapshot may be an exported snapshot of the replicated snapshot on the destination. *Id.*, ¶[0120].

658. In sum, Birse and Federwisch disclose storing said indexing results on a shared storage unit.

9. Claim 21: The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the compute nodes and said first storage unit and respective second storage units.

659. *See* claim limitations [19.0]-[19.4] (§XII.D.7 above) in this Ground 4, and claim limitation [12] (§XII.C.7 above) in Ground 3 above.

10. Claim 22: The method as recited in claim 19 wherein said first storage unit is read-only.

660. *See* claim limitations [19.0]-[19.4] (§XII.D.7 above) in this Ground 4, and claim limitation [13] (§XII.C.8 above) in Ground 3 above.

11. Claim 23

a. [23.0] A computer-readable storage medium having instructions stored thereon that, in response to execution by at least one computing device, cause the at least one computing device to:

661. *See* claim limitation [19.0] in this Ground 4 (§XII.D.7.a above).

b. [23.1]-[23.3] receive data blocks of a file system, said data blocks comprising a root image portion and leaf image portion, said leaf image portion comprising only additional data blocks not previously contained in said root image portion and changes made by said first compute node to the blocks of said root image, wherein said leaf image portion does not include blocks of said root image that are unchanged by said first compute node, wherein said file system is the result of merging said root image portion and said leaf image portion together at the block-level;

662. *See* claim limitations [19.2] (§XII.D.7.c above) and [19.3] (§XII.D.7.d above) in this Ground 4.

c. [23.4] index said root image portion; and

663. *See* claim limitations [19.4] in this Ground 4 (§XII.D.7.e above).

d. [23.4] provide the results of said indexing to a second compute node, wherein said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node.

664. *See* claim limitations [19.4] (§XII.D.7.f above) and [14.3]-[14.4] (§XII.D.1.d above) in this Ground 4.

12. Claim 24: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to store said results of said indexing on a shared storage unit accessible by said second compute node.

665. *See* claim limitations [23.0]-[23.4] (§XII.D.11 above) and [20] (§XII.D.8 above) in this Ground 4.

13. Claim 25: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion.

666. *See* claim limitation [14.4] in this Ground 4 (§XII.D.1.d above).

667. Moreover, a POSITA would have known that indexing is an old and common technique used well before 2004. Specifically, a POSITA would have understood that when some data changes on a disk or file system, it is inefficient to scan and re-index an entire system, only to reprocess unchanged data; instead, it is more efficient to index only files that have changed. This is easily achieved because storage and file systems include timestamps that denote when the files/data were last modified. *See* §VII.B above; *see also* §VII.F above.

668. A POSITA would have been motivated to combine Birse and Federwisch for the reasons discussed in §XII.D.2 above.

669. *See* claim limitations [23.0]-[23.4] in this Ground 4 (§XII.D.11 above).

670. Birse and Federwisch further disclose the instructions further causing the at least one computing device to index said leaf image portion.

671. Federwisch compares the indexes to file's block numbers between two snapshots to identify whether data has been changed between the snapshots (as would be accomplished using the CoW techniques discussed in §VII.E above).

672. "The existence of a changed volume block number at a given offset in the index between the base snapshot inode file and incremental snapshot inode file generally indicates that one or more of the underlying inodes and files to which the inodes point have been changed." EX1008 (Federwisch), ¶[0071].

673. Federwisch scans these indexes to identify unchanged blocks. "A scanner 820 searches the index for changed base/incremental inode file snapshot blocks, comparing volume block numbers or another identifier. … Hence, scanned blocks at the same index in both snapshots can be efficiently bypassed, reducing the scan time." EX1008 (Federwisch), ¶[0072].

674. A POSITA would have understood these disclosures to mean that in Federwisch, each snapshot (corresponding to the '844 Patent's "leaf image") has its own index of file blocks. "More specifically, each entry in the inode map 1400

contains an entry for each inode on the source snapshot. Each inode entry 1402 in the map is indexed and accessed via the source inode number (1404)." EX1008 (Federwisch), ¶[0109].

675. In sum, Birse and Federwisch disclose the instructions further causing the at least one computing device to index said leaf image portion.

14. Claim 26: The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.

676. *See* claims limitation [14.4] (§XII.D.1.d above) and claim limitation [25] (§XII.D.13 above) in this Ground 4.

677. Moreover, a POSITA would have known that indexing is an old and common technique used well before 2004. Specifically, a POSITA would have understood that when some data changes on a disk or file system, it is inefficient to scan and re-index an entire system, only to reprocess unchanged files; instead, it is more efficient to index only files that have changed. This is easily achieved because storage and file systems include timestamps that denote when the files were last modified. Lastly, when some data changes, for example in any form of a simple database (such as an index), a POSITA would have understood that the new index data would be easily merged with older data: this would be useful to purge now-stale index entries. *See* §VII.B above; s*ee also* §VII.F above.

678. A POSITA would have been motivated to combine Birse and Federwisch for the reasons discussed in §XII.D.2 above.

679. Federwisch even discloses a form of re-indexing of incremental snapshots (which the '844 Patent calls "leaf images"). In one example, "the Destination Snapshot A (2002) is now prepared to transfer changes so as to establish a mirror in Destination Snapshot B (2004). … In this case, both the new source and the new destination system have their own inode maps A and B (2012 and 2014), indexed off the old source 2001, and referencing the inodes in their respective trees. … It then treats the new source as the appropriate map index for the new associated inode map 2018. In the associated map, it stores the new source generation number for the new source index inode number, with each index entry also associated with/mapped to the new destination inode/generation number extracted from the inode map B (2014)." EX1008 (Federwisch), ¶[0137]. *See also* §X.C above.

680. *See* claim limitation [25] in this Ground 4 (§XII.D.13 above).

681. Birse and Federwisch further disclose the instructions further causing the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.

682. When snapshots in Federwisch are copied from a source to a destination, although the data has not changed, the snapshot's block locations may have changed. Thus, Federwisch updates the destination snapshot's index maps, a form of re-indexing.

683. In sum, Birse and Federwisch disclose the instructions further causing the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.

15. Claim 27: The computer-readable storage medium of claim 23 wherein said root image portion is read-only.

684. *See* claim limitations [23.0]-[23.4] (§XII.D.11 above) in this Ground 4, claim limitation [13] (§XII.C.8 above) in Ground 3 above.

XIII. SECONDARY CONSIDERATIONS

685. I have seen no evidence in either the '844 Patent's prosecution history or elsewhere supporting any secondary considerations arguments, or evidence of nexus of such alleged evidence to the challenged claims. To the extent patent owner or its expert asserts the existence of any secondary considerations in its responses, I reserve the right to address any such evidence.

XIV. CONCLUSION

686. The findings and opinions set forth in this declaration are based on my work and examinations to date.

687. I may continue my examinations. I may also receive additional documentation and other factual evidence over the course of this IPR that will allow me to supplement and/or refine my opinions. I reserve the right to add to, alter, or delete my opinions and my declaration upon discovery of any additional information. I reserve the right to make such changes as may be deemed necessary.

688. In signing this declaration, I recognize that the declaration will be filed as evidence in an IPR before the PTAB. I also recognize that I may be subject to cross-examination in the case and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

689. 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 are punishable by fine or imprisonment, or both, under §1001 of Title 18 of the United States Code.

Date: April 25, 2025

Erez Zadok, Ph.D.