IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

VALTRUS INNOVATIONS LTD. and KEY PATENT INNOVATIONS LTD.,	§	Case No. 2:25-cv-
	§	**JURY TRIAL DEMANDED**
Plaintiffs,	§	
v.	§	
FEDEX CORPORATION and FEDEX OFFICE AND PRINT SERVICES, INC.,	§	
Defendants.	§	

COMPLAINT FOR PATENT INFRINGEMENT

Plaintiffs Valtrus Innovations Ltd. ("Valtrus") and Key Patent Innovations Ltd. ("KPI") (collectively, "Plaintiffs") for their Complaint against Defendants FedEx Corporation and FedEx Office and Print Services, Inc. (collectively, "FedEx" or "Defendants") for patent infringement allege as follows:

THE PARTIES

1. Valtrus is the successor-in-interest to a substantial patent portfolio created by Hewlett Packard Enterprise and its predecessor, subsidiary, and affiliate companies, including Hewlett-Packard Development Company, L.P. (collectively, "HPE"). Valtrus is an Irish entity duly organized and existing under the laws of Ireland. The address of the registered office of Valtrus is: The Glasshouses GH2, 92 Georges Street Lower, Dun Laoghaire, Dublin A96 VR66, Ireland. HPE's worldwide corporate headquarters is located in Houston, Texas. One of HPE's primary U.S. facilities is located in Plano, Texas.

FEDEX 1041
Fedex Corp. v. Valtrus Innovations Ltd.
IPR2025-01491

2. KPI is the beneficiary of a trust pursuant to which Valtrus owns, holds, and asserts the Patents-in-Suit as defined in Paragraph 20. KPI is an Irish entity duly organized and existing under the laws of Ireland. The address of the registered office of KPI is: The Glasshouses GH2, 92 Georges Street Lower, Dun Laoghaire, Dublin A96 VR66, Ireland.

3. On information and belief, Defendant FedEx Corporation is a company organized under the laws of the State of Delaware, with a place of business at 942 South Shady Grove Road, Memphis, TN 38120. On information and belief, Defendant FedEx Corporation may be served through its registered agent for service, CT Corporation System located at 1999 Bryan Street, Suite 900, Dallas, TX 75201-3136. On information and belief, Defendant FedEx Corporation does business in Texas, directly or through intermediaries, and offers its products and/or services, including those accused herein of infringement, to customers and potential customers located in Texas, including in the Judicial District of the Eastern District of Texas.

4. On information and belief, Defendant FedEx Office and Print Services, Inc. is a company organized under the laws of the State of Delaware, with its corporate headquarters located at 7900 Legacy Drive, Plano, TX 75024.

Corporate Headquarters

7900 Legacy Drive

Plano, TX 75024[1]

On information and belief, Defendant FedEx Office and Print Services, Inc. may be served through its registered agent for service, CT Corporation System located at 1999 Bryan Street, Suite 900, Dallas, TX 75201-3136. On information and belief, Defendant FedEx Office and Print Services,

[1] https://www.office.fedex.com/?srsltid=AfmBOoohPlnZSz4frxMjdUUOj-iFtoomxvrQ1mtnCPeHWv5m_p3mHR0t

Inc. is a wholly-owned subsidiary of Defendant FedEx Corporation and carries out the business of its parent throughout the United States and Texas. On information and belief, Defendant FedEx Office and Print Services, Inc. does business in Texas, directly or through intermediaries, and offers its products and/or services, including those accused herein of infringement, to customers and potential customers located in Texas, including in the Judicial District of the Eastern District of Texas.

5. FedEx controls, participates in the commission of, and has a direct financial interest in the infringing acts set forth herein.

JURISDICTION AND VENUE

6. This is an action for patent infringement arising under the patent laws of the United States, 35 U.S.C. §§ 1, *et seq*. This Court has jurisdiction over this action pursuant to 28 U.S.C. §§ 1331 and 1338(a).

7. This Court has personal jurisdiction over Defendants. Defendants regularly conduct business and have committed acts of patent infringement and/or have induced acts of patent infringement by others in this Judicial District and/or have contributed to patent infringement by others in this Judicial District, the State of Texas, and elsewhere in the United States. Defendants, directly and/or through subsidiaries or intermediaries, have committed and/or continue to commit acts of infringement in this District by, among other things, making, using, importing, offering to sell, and/or selling products that infringe the Patents-in-Suit as defined in Paragraph 17.

8. This Court has general jurisdiction over Defendants due to their continuous and systematic contacts with the State of Texas, including through their operation of physical offices in the State of Texas, either directly or through authorized third parties, including this District. These locations include drop-off locations, locations with shipping assistance, locations with

package holding and/or shipping, or a combination thereof. These physical locations include at least the following: 1300 E Whaley Steet, Longview, TX 75601; 916 W Loop 281, Longview, TX 75604; 3320 Robertson Road, Tyler, TX 75701; 4522 S Broadway Avenue, Tyler, TX 75703; 1200 Capital Avenue, Plano, TX 75074; 901 N. Central Expy, Suite 200, Plano, TX 75075; 1415 E Renner Road, Suite 230, Richardson, TX 75082; and 1565 N Central Expy, Suite 100, Richardson, TX 75080. Additionally, this Court has general jurisdiction over Defendants due to their targeting of Texas residents with the https://www.fedex.com/en-us/home.html and https://www.office.fedex.com/ websites, their ownership and/or lease of land in the State of Texas, and other business activities throughout the State of Texas.

9. Defendants are subject to the specific personal jurisdiction of this Court because Plaintiffs' patent infringement claims against Defendants specifically arise from Defendants' acts of infringement in the State of Texas. These acts of infringement include directing and controlling the operation of the https://www.fedex.com/en-us/home.html and https://www.office.fedex.com/ websites, using the patented inventions, and specifically targeting residents of Texas with these and other websites to further the sale of products and services to those customers online in this District. On information and belief, Defendants operate and/or contract with third parties to operate various locations through the FedEx Authorized ShipCentre program[2], including in the State of Texas.

10. Venue is proper in this District under 28 U.S.C. §§ 1391(b)-(d) and 1400(b). Defendants are registered to do business in the State of Texas, have offices in the State of Texas, have transacted and continue to transact business in the Eastern District of Texas, and have

[2]https://www.fedex.com/en-ca/retail-locations/pickup-dropoff/joinshipcentre.html#:~:text=FASC%20Application,GoFedEx

committed and continue to commit acts of direct and indirect infringement in the Eastern District of Texas.

11. Defendants are subject to this Court's jurisdiction pursuant to due process and/or the Texas Long Arm Statute due at least to their substantial business in this State and Judicial District, including (a) at least part of their past infringing activities, (b) regularly doing or soliciting business in Texas, and/or (c) engaging in persistent conduct and/or deriving substantial revenue from goods and services provided to customers in Texas.

PATENTS-IN-SUIT

12. On February 5, 2013, the United States Patent and Trademark Office duly and legally issued U.S. Patent No. 8,370,416 (the "'416 Patent") entitled "Compatibility Enforcement in Clustered Computing Systems." A true and correct copy of the '416 Patent is attached hereto as Exhibit A.

13. On March 8, 2011, the United States Patent and Trademark Office duly and legally issued U.S. Patent No. 7,904,686 (the "'686 Patent") entitled "Data Security for Use with a File System." A true and correct copy of the '686 Patent is attached hereto as Exhibit B.

14. On December 29, 2009, the United States Patent and Trademark Office duly and legally issued U.S. Patent No. 7,640,332 (the "'332 Patent") entitled "System and Method for Hot Deployment/Redeployment in Grid Computing Environment." A true and correct copy of the '332 Patent is attached hereto as Exhibit C.

15. On October 10, 2006, the United States Patent and Trademark Office duly and legally issued U.S. Patent No. 7,120,832 (the "'832 Patent") entitled "Storage Device Performance Monitor." A true and correct copy of the '832 Patent is attached hereto as Exhibit D.

16. Plaintiffs are the sole and exclusive owner of all legal right, title, and interest in the

'416 Patent, the '686 Patent, the '332 Patent, and the '832 Patent (collectively, the "Patents-in-Suit" or "Asserted Patents"), and hold the exclusive right to take all actions necessary to enforce their rights to the Patents-in-Suit, including the filing of this patent infringement lawsuit. Plaintiffs also have the right to recover all damages for past, present, and future infringement of the Patents-in-Suit and to seek injunctive relief as appropriate under the law.

FACTUAL ALLEGATIONS

17. The '416 Patent generally relates to a system and method for a network comprising electronic devices associated with user profiles to allow migration of the user profiles from one electronic device to another. The technology described in the '416 Patent was developed by David J. Hoover, Bernard A. Reed, III, Marjorie Krueger, and Douglas L. Voigt at Hewlett-Packard Development Company, L.P. For example, the '416 Patent is infringed through FedEx's use of Kubernetes for cluster creation.

18. The '686 Patent generally relates to a system and method for providing data security for use with a file system. The technology described in the '686 Patent was developed by Ranganath G. Iyengar at Hewlett-Packard Development Company, L.P. For example, the '686 Patent is infringed by FedEx's Apache Software Foundation's Hadoop Distributed File System ("HDFS").

19. The '332 Patent generally relates to technology for hot deployment and/or redeployment in a grid computing environment, where the grid computing environment adds a new version of an application from a repository server, determining which grid node is running the application affected by the new version of the software, notifying the client application manager associated with one or more of the affected grid nodes, and hot deploying/redeploying the new version of the application to the affected grid nodes. The technology described in the '332

Patent was developed by Sanjay Dahiya at Hewlett-Packard Development Company, L.P. For example, the '332 Patent is infringed by FedEx's use of Kubernetes for rolling updates.

20. The '832 Patent generally relates to technology for monitoring performance of a storage device by intercepting communications between a computer system and said storage device, analyzing the intercepted communications, and reallocating at least some of said data on said storage device to enhance the performance of the storage device. The technology described in the '832 Patent was developed by Kevin Collins and Michael P. Fleischmann at Hewlett-Packard Development Company, L.P. For example, the '823 Patent was infringed by FedEx's use of Apache Kafka Streams using RocksDB Universal Compaction, RocksDB Universal Compaction, Apache Cassandra, and Apache HBase.

21. FedEx has infringed the '832 Patent by making, using, selling, offering to sell, and/or importing, and by actively inducing others to make, use, sell, offer to sell, and/or import the infringing products identified above.

22. FedEx has infringed and is continuing to infringe the '416 Patent, the '686 Patent, and the '332 Patent by making, using, selling, offering to sell, and/or importing, by actively inducing others to make, use, sell, offer to sell, and/or import, and/or contributing to other's making, using, selling, offering to sell, and/or importing of the infringing products identified above.

COUNT I
(Infringement of the '416 Patent)

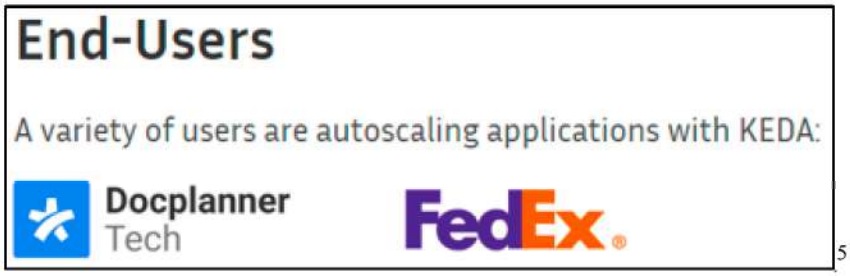
23. Paragraphs 1 through 22 are incorporated by reference as if fully set forth herein.

24. Plaintiffs have not licensed or otherwise authorized Defendants to make, use, offer for sale, sell, or import any products that embody the inventions of the '416 Patent.

25. Defendants have directly infringed and continue to directly infringe the '416 Patent, either literally or under the doctrine of equivalents, without authority and in violation of 35 U.S.C. § 271, by making, using, offering to sell, selling, and/or importing into the United States products that satisfy each and every limitation of one or more claims of the '416 Patent. Such products include, but are not limited to, Kubernetes.

26. For example, Defendants have directly infringed and continue to directly infringe at least claim 1 of the '416 Patent by making, using, offering to sell, selling, and/or importing into the United States products that include Kubernetes.

27. On information and belief, FedEx has used Kubernetes in the U.S. since at least 2019..[3] On information and belief, FedEx had developers build or refactor FedEx applications on a PaaS (Platform as a Service) system. The PaaS system enables engineers to shuttle applications between Google Cloud Platform and internal datacenters by using container software to make the applications portable and the Kubernetes technology to orchestrate the movement of containers between clouds..[4]



[5]

[3] https://www.cio.com/article/217625/fedex-cloud-dojo-accelerates-digital-transformation.html
[4] https://www.cio.com/article/217625/fedex-cloud-dojo-accelerates-digital-transformation.html. See also, e.g., https://clavent.com/top-devops-trends-that-will-matter-the-most-in-2020/ (discussing FedEx's Cloud Dojo). See also, e.g., https://www.linkedin.com/in/monikamevenkamp/; (showing use of Kubernetes); https://www.linkedin.com/in/asaf-peleg-007/ (showing use of both Kubernetes and Kafka)
[5] https://keda.sh/community/#end-users

Kubernetes Workloads Autoscaling with KEDA

KEDA (Kubernetes Event Driven Autoscaling) is an Open Source tool with the backing of large companies (e.g. Microsoft, RedHat, IBM, etc.), and it is also a CNCF (Cloud Native Computing Foundation) graduate project. It is used by a large variety of companies, big and small: Fedex, Reddit, Xbox, Cisco, etc.

***KEDA** is a Kubernetes-based Event Driven Autoscaler. With KEDA, you can drive the scaling of any container in Kubernetes based on the number of events needing to be processed.*

***KEDA** is a single-purpose and lightweight component that can be added into any Kubernetes cluster. KEDA works alongside standard Kubernetes components like the Horizontal Pod Autoscaler and can extend functionality without overwriting or duplication. With KEDA you can explicitly map the apps you want to use event-driven scale, with other apps continuing to function. This makes KEDA a flexible and safe option to run alongside any number of other Kubernetes applications or frameworks.*[6]

Delivering at the Nexus of the Digital & Physical Worlds: VMware & FedEx

"At FedEx, we stand at the nexus of the digital and physical worlds," said FedEx CIO Rob Carter as he shared FedEx's success story of using VMware services to quickly and efficiently scale to meet the intense demands of delivering the massive rollout of the life-saving Coronavirus vaccines as well as the surge in e-commerce deliveries due to stay-at-home requirements.

[6] https://igboie.medium.com/kubernetes-workloads-autoscaling-with-keda-ccba62b64552

FedEx Senior Enterprise Architect, Matt Lum, called VMware Tanzu "critical" to the company's "day-to-day", especially with FedEx's demanding testing practices. VMware Tanzu, a portfolio of products and services that helps enterprises build, run and manage applications on Kubernetes ("an open-source container orchestration platform") is a service FedEx lauds as helping the company not only in its day-to-day operations, but also in giving it a greater ability to be agile, innovative, and efficient in meeting new demands with new applications.

Lum, credited VMware Tanzu with helping FedEx scale up to meet the demands of delivering the Covid vaccine roll-out as FedEx introduced new applications to more closely track deliveries by providing a trusted infrastructure.[7]

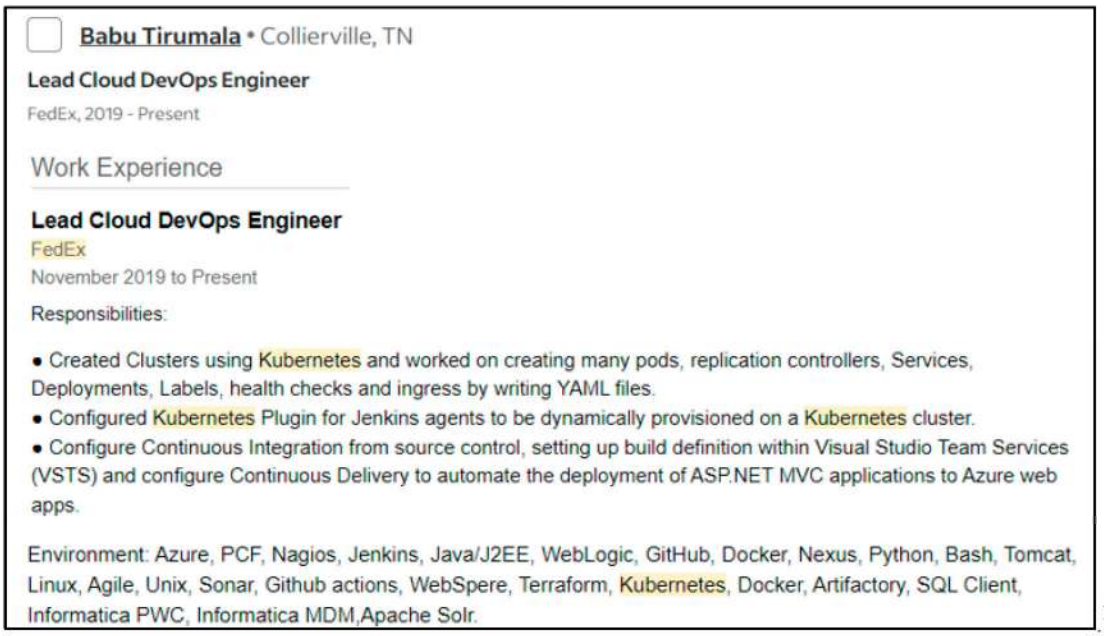


[8]

[7] https://www.linkedin.com/pulse/delivering-nexus-digital-physical-worlds-vmware-fedex-glen-gilmore/
[8] https://www.linkedin.com/in/kcnagareddy/

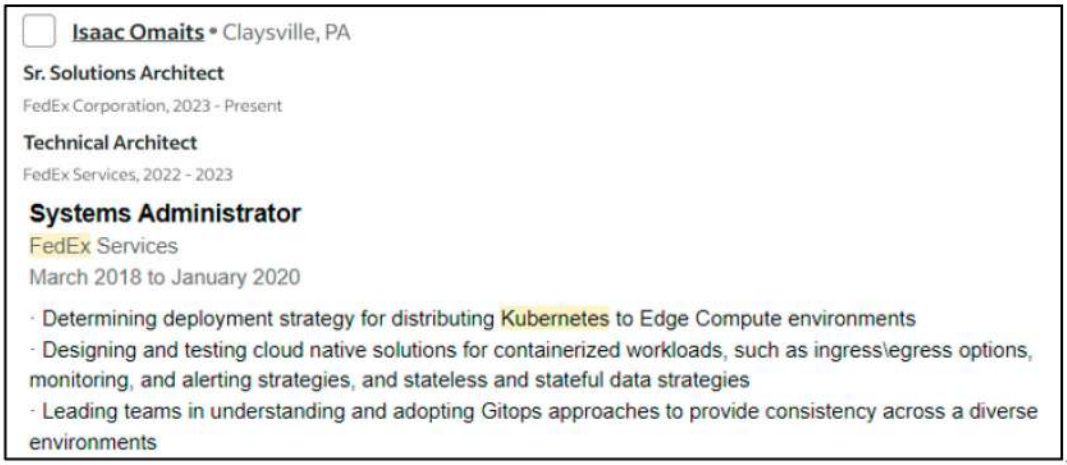


[9]



[10]

[9] https://www.linkedin.com/in/joshua-herndon-60890337/
[10] resumes.indeed.com – Smart Search (Login Required).



Isaac Omaits • Claysville, PA

Sr. Solutions Architect

FedEx Corporation, 2023 - Present

Technical Architect

FedEx Services, 2022 - 2023

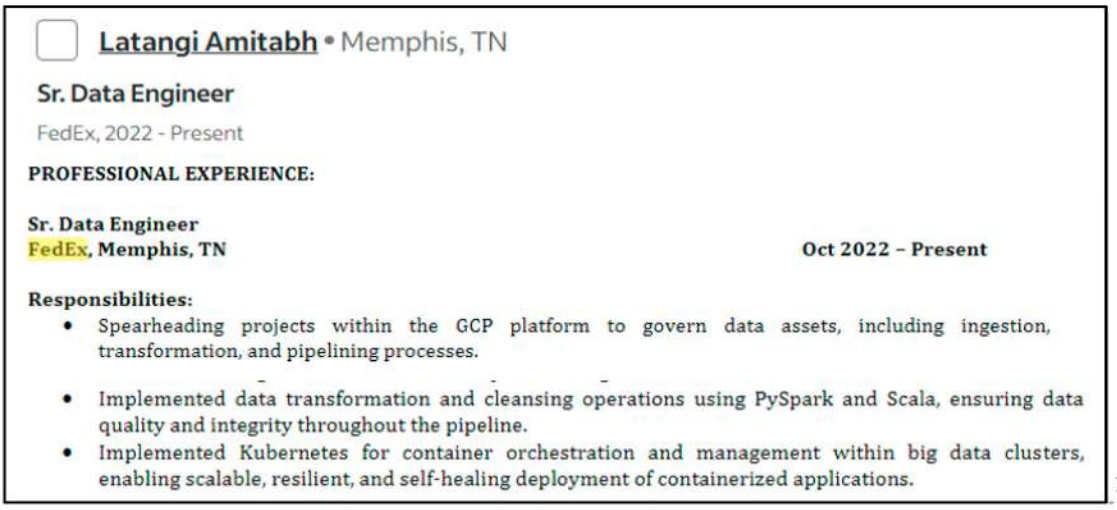
Systems Administrator

FedEx Services

March 2018 to January 2020

- Determining deployment strategy for distributing Kubernetes to Edge Compute environments
- Designing and testing cloud native solutions for containerized workloads, such as ingress\egress options, monitoring, and alerting strategies, and stateless and stateful data strategies
- Leading teams in understanding and adopting Gitops approaches to provide consistency across a diverse environments

[11]



Latangi Amitabh • Memphis, TN

Sr. Data Engineer

FedEx, 2022 - Present

PROFESSIONAL EXPERIENCE:

Sr. Data Engineer
FedEx, Memphis, TN **Oct 2022 – Present**

Responsibilities:

- Spearheading projects within the GCP platform to govern data assets, including ingestion, transformation, and pipelining processes.
- Implemented data transformation and cleansing operations using PySpark and Scala, ensuring data quality and integrity throughout the pipeline.
- Implemented Kubernetes for container orchestration and management within big data clusters, enabling scalable, resilient, and self-healing deployment of containerized applications.

[12]

[11] resumes.indeed.com – Smart Search (Login Required).
[12] resumes.indeed.com – Smart Search (Login Required).



[13]

[13] https://newsroom.fedex.com/newsroom/global/fedex-completes-acquisition-of-shoprunner-expanding-its-e-commerce-capabilities



ShopRunner
3 yrs 11 mos

Principal Engineer
Mar 2021 - Oct 2021 · 8 mos
Conshohocken, Pennsylvania, United States

Designed, managed and programmed the companies internal customer support app that included a React UI, Kotlin Backend and integration with Zendesk

Contributed to the rebuilding of the order management system moving it to a distributed system with a Kafka backbone.

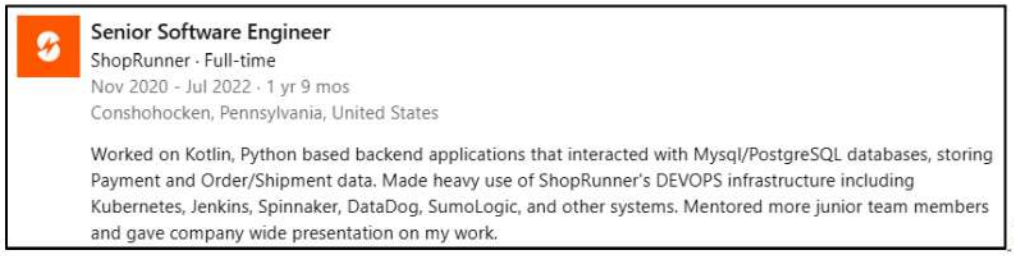
Lead Engineer In Payments
Aug 2019 - Mar 2021 · 1 yr 8 mos

Grew and mentored team of engineers
Developed React Native digital wallet e-commerce application
Led legacy application migration from Python to Kotlin

Lead Engineer In CloudOps
Mar 2019 - Aug 2019 · 6 mos

Migrated container infrastructure to Kubernetes
Created templates for k8s deployments used across the organization
Advocated & built self service model to deploy infrastructure using Terraform

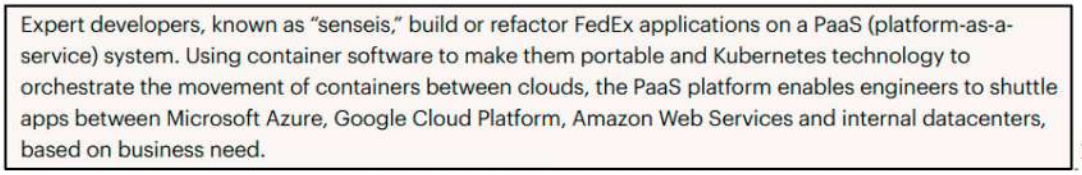
[14]



Senior Software Engineer
ShopRunner · Full-time
Nov 2020 - Jul 2022 · 1 yr 9 mos
Conshohocken, Pennsylvania, United States

Worked on Kotlin, Python based backend applications that interacted with Mysql/PostgreSQL databases, storing Payment and Order/Shipment data. Made heavy use of ShopRunner's DEVOPS infrastructure including Kubernetes, Jenkins, Spinnaker, DataDog, SumoLogic, and other systems. Mentored more junior team members and gave company wide presentation on my work.

[15]



Expert developers, known as "senseis," build or refactor FedEx applications on a PaaS (platform-as-a-service) system. Using container software to make them portable and Kubernetes technology to orchestrate the movement of containers between clouds, the PaaS platform enables engineers to shuttle apps between Microsoft Azure, Google Cloud Platform, Amazon Web Services and internal datacenters, based on business need.

[16]

28. Kubernetes performs a method of creating a clustered computing system. For example, in Kubernetes, the Kubeadm tool is used to create clusters.

Kubeadm

[14] https://www.linkedin.com/in/asaf-peleg-007/details/experience/
[15] https://www.linkedin.com/in/monikamevenkamp/
[16] https://www.cio.com/article/217625/fedex-cloud-dojo-accelerates-digital-transformation.html

Kubeadm is a tool built to provide `kubeadm init` and `kubeadm join` as best-practice "fast paths" for creating Kubernetes clusters.[17]

Creating a cluster with kubeadm

Using `kubeadm`, you can create a minimum viable Kubernetes cluster that conforms to best practices. In fact, you can use `kubeadm` to set up a cluster that will pass the Kubernetes Conformance tests. `kubeadm` also supports other cluster lifecycle functions, such as bootstrap tokens and cluster upgrades.[18]

29. Kubernetes performs the step of storing license information for a computing cluster in a memory module associated with a computing node wherein the licensing information includes a bundle-type parameter that identifies a size of the cluster and at least one additional bundle-type characteristic of the cluster. For example, Kubeadm stores information including the number of worker nodes and number of control plane, and the characteristics of nodes.

Creating a cluster with kubeadm

Using `kubeadm`, you can create a minimum viable Kubernetes cluster that conforms to best practices. In fact, you can use `kubeadm` to set up a cluster that will pass the Kubernetes Conformance tests. `kubeadm` also supports other cluster lifecycle functions, such as bootstrap tokens and cluster upgrades.[19]

This page explains two different approaches to setting up a highly available Kubernetes cluster using kubeadm:

[17] https://kubernetes.io/docs/reference/setup-tools/kubeadm/
[18] https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/
[19] https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/

- Three or more machines that meet kubeadm's minimum requirements for the control-plane nodes. Having an odd number of control plane nodes can help with leader selection in the case of machine or zone failure.
  - including a container runtime, already set up and working
- Three or more machines that meet kubeadm's minimum requirements for the workers
  - including a container runtime, already set up and working [20]

kubeadm's minimum requirements

- Unique hostname, MAC address, and product_uuid for every node. See here for more details. [21]

for handling container operations, such as containerd or CRI-O. A Kubernetes cluster that handles production traffic should have a minimum of three nodes because if one node goes down, both an etcd member and a control plane instance are lost, and redundancy is compromised. You can mitigate this risk by adding more control plane nodes. [22]

1. (Recommended) If you have plans to upgrade this single control-plane `kubeadm` cluster to high availability you should specify the `--control-plane-endpoint` to set the shared endpoint for all control-plane nodes. Such an endpoint can be either a DNS name or an IP address of a load-balancer. [23]

30. Kubernetes performs the steps of initializing the computing cluster in a first computing node and adding one or more available computing nodes to the computing cluster. For example, the "Kubeadm init" command initiates the cluster on a control plane node. Other nodes join this initialized cluster, in turn.

Initializing your control-plane node

[20] https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/
[21] https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/
[22] https://kubernetes.io/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/
[23] https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/

To initialize the control-plane node run:

```
kubeadm init <args>
```

`kubeadm init` first runs a series of prechecks to ensure that the machine is ready to run Kubernetes. These prechecks expose warnings and exit on errors. `kubeadm init` then downloads and installs the cluster control plane components. This may take several minutes. After it finishes you should see:

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a Pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  /docs/concepts/cluster-administration/addons/

You can now join any number of machines by running the following on each node
as root:
```

```
kubeadm join <control-plane-host>:<control-plane-port> --token <token>
```

```
--discovery-token-ca-cert-hash sha256:<hash>
```

The nodes are where your workloads (containers and Pods, etc) run. To add new nodes to your cluster do the following for each machine:

- Run the command that was output by `kubeadm init`. For example:

[24]

31. Kubernetes performs the step of activating the computing cluster when the computing cluster includes a number of nodes that comply with the bundle-type parameter. For

[24] https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/

example, when all specified worker nodes are ready, and the control-plane joined and configured, the formed cluster is activated.

- Three or more machines that meet kubeadm's minimum requirements for the control-plane nodes. Having an odd number of control plane nodes can help with leader selection in the case of machine or zone failure.
  - including a container runtime, already set up and working
- Three or more machines that meet kubeadm's minimum requirements for the workers
  - including a container runtime, already set up and working

[25]

Node conformance test is a containerized test framework that provides a system verification and functionality test for a node. The test validates whether the node meets the minimum requirements for Kubernetes; a node that passes the test is qualified to join a Kubernetes cluster. [26]

A few seconds later, you should notice this node in the output from `kubectl get nodes` when run on the control-plane node.

```
[preflight] Running pre-flight checks

... (log output of join workflow) ...

Node join complete:
* Certificate signing request sent to control-plane and response
  received.
* Kubelet informed of new secure connection details.

Run 'kubectl get nodes' on control-plane to see this machine join.
```

1. List the nodes in your cluster, along with their labels:

```
kubectl get nodes --show-labels
```

```
NAME      STATUS    ROLES    AGE     VERSION        LABELS
worker0   Ready     <none>   1d      v1.13.0        ...,kubernetes.io/hostname=worker0
worker1   Ready     <none>   1d      v1.13.0        ...,kubernetes.io/hostname=worker1
worker2   Ready     <none>   1d      v1.13.0        ...,kubernetes.io/hostname=worker2
```

[27]

32. Defendants directly infringe the '416 Patent through their implementation and use of Kubernetes for cluster creation, performing at least the steps of claim 1.

[25] https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/
[26] https://kubernetes.io/docs/setup/best-practices/node-conformance/
[27] https://kubernetes.io/docs/tasks/configure-pod-container/assign-pods-nodes/

33. Defendants have indirectly infringed and continue to indirectly infringe one or more claims of the '416 Patent by knowingly and intentionally inducing others, including FedEx customers and end-users, to directly infringe, either literally or under the doctrine of equivalents, by making, using, offering to sell, selling, and/or importing into the United States products that include the infringing technology.

34. Defendants, with knowledge that these products, or the use thereof, infringe the '416 Patent at least as of October 16, 2024,[28] knowingly and intentionally induced, and continue to knowingly and intentionally induce, direct infringement of the '416 Patent by providing these products to end-users for use in an infringing manner. Additionally, on information and belief, Defendants have adopted a policy of not reviewing the patents of others, including specifically those related to Defendants' specific industry, thereby remaining willfully blind to the '416 Patent at least as early as the issuance of the Patents-in-Suit.

35. Defendants have induced infringement by others, including end-users, with the intent to cause infringing acts by others or, in the alternative, with the belief that there was a high probability that others, including end-users, infringe the '416 Patent, but while remaining willfully blind to the infringement. Defendants have induced, and continue to induce, infringement by their customers and end-users by supplying them with instructions on how to operate the infringing technology in an infringing manner, while also making publicly available information on the

[28] Plaintiffs sent Defendant FedEx Corporation correspondence on October 16, 2024, providing notice of infringement of the '416 Patent. Therefore, Defendants had knowledge of the '416 Patent as of at least October 2024

infringing technology via Defendants' website, product literature and packaging, and other publications..[29]

36. Plaintiffs have suffered damages as a result of Defendants' direct and indirect infringement of the '416 Patent in an amount to be proven at trial.

37. Plaintiffs have suffered, and will continue to suffer, irreparable harm as a result of Defendants' infringement of the '416 Patent, for which there is no adequate remedy at law, unless Defendants' infringement is enjoined by this Court.

38. Defendants have committed and continue to commit acts of infringement that Defendants actually knew or should have known constituted an unjustifiably high risk of infringement of at least one valid and enforceable claim of the '416 Patent. Defendants' direct and indirect infringement of the '416 Patent has been and continue to be willful, intentional, deliberate, and/or in conscious disregard of rights under the patent. Plaintiffs are entitled to an award of treble damages, reasonable attorney fees, and costs in bringing this action.

COUNT II
(Infringement of the '686 Patent)

39. Paragraphs 1 through 22 are incorporated by reference as if fully set forth herein.

40. Plaintiffs have not licensed or otherwise authorized Defendants to make, use, offer for sale, sell, or import any products that embody the inventions of the '686 Patent.

41. Defendants have infringed, and continue to directly infringe, the '686 Patent, either literally or under the doctrine of equivalents, without authority and in violation of 35 U.S.C. § 271, by making, using, offering to sell, selling, and/or importing into the United States products that

[29] *See, e.g.,* https://www.fedex.com/en-us/developer/get-support.html#:~:text=1.800.GoFedEx%201.800.463.3339%20(,say%20%E2%80%9CInternational%20Services.%E2%80%9D); https://www.fedex.com/en-us/customer-support.html

satisfy each and every limitation of one or more claims of the '686 Patent. Such products include HDFS.

42. On information and belief, FedEx has used Apache Hadoop in the U.S. since at least 2019.

FedEx.

Data Scientist - All Levels

Company: FedEx Dataworks

Category: Professional

Employment Type: Full Time

Worker Sub-Type: Regular Remote Worker

Remote: Yes

Location:

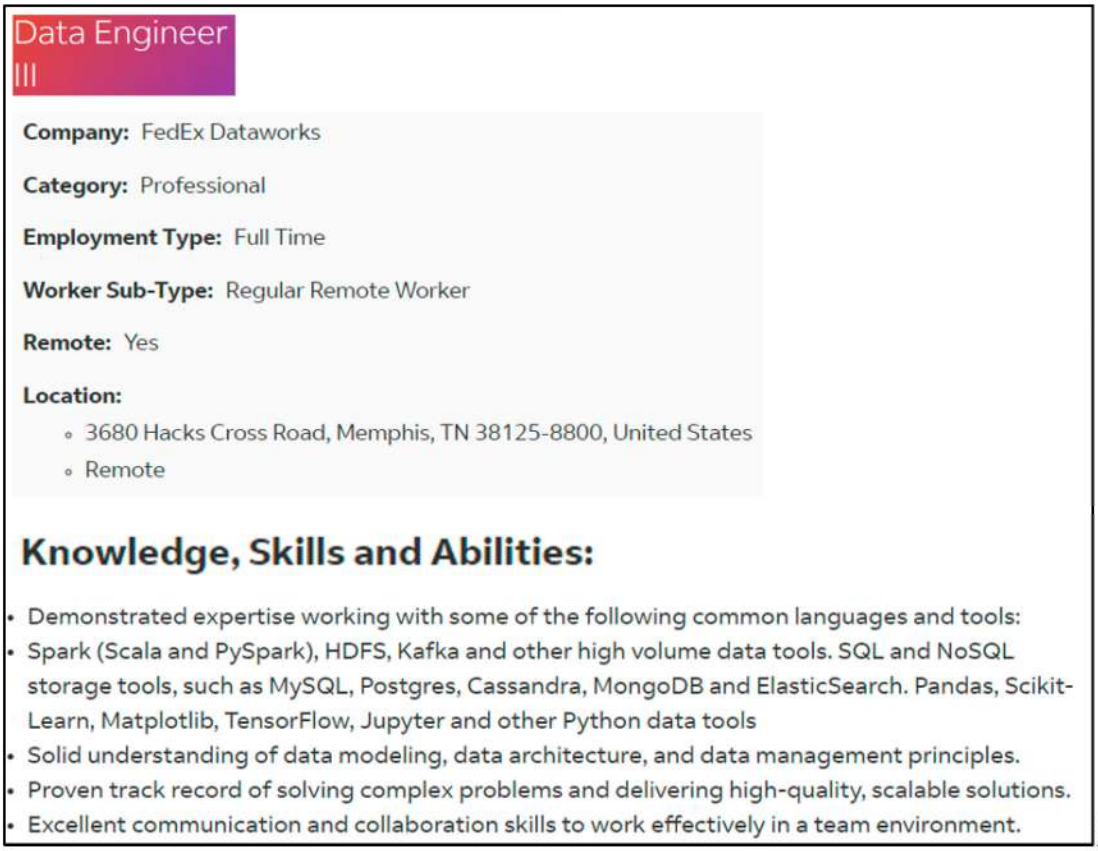
- 3680 Hacks Cross Road, Memphis, TN 38125-8800, United States
- Remote

Skills/Abilities:

- Technical background in computer science, data science, machine learning, artificial intelligence, statistics or other quantitative and computational science
- A compelling track record of Data Science / Engineer expertise, designing and deploying large scale technical solutions, which deliver tangible, ongoing value
- Direct experience having built and deployed robust, complex production systems through Cloud technologies that implement modern, data scientific methods at scale
- SKLearn, XGBoost, Tensorflow, Pytorch, MLlib and other core ML frameworks
- Python, Scala, R, C++, Java and other modern programming languages
- MLFlow, Databricks, Spark, Kafka, Hive, Hadoop and other data tools and frameworks
- CPLEX, Gurobi and other similar optimization modeling packages

[30]

[30] https://careers.fedex.com/data-scientist-all-levels/job/P25-101260-3?utm_source=CareerSiteOrganic



Data Engineer III

Company: FedEx Dataworks

Category: Professional

Employment Type: Full Time

Worker Sub-Type: Regular Remote Worker

Remote: Yes

Location:

- 3680 Hacks Cross Road, Memphis, TN 38125-8800, United States
- Remote

Knowledge, Skills and Abilities:

- Demonstrated expertise working with some of the following common languages and tools:
- Spark (Scala and PySpark), HDFS, Kafka and other high volume data tools. SQL and NoSQL storage tools, such as MySQL, Postgres, Cassandra, MongoDB and ElasticSearch. Pandas, Scikit-Learn, Matplotlib, TensorFlow, Jupyter and other Python data tools
- Solid understanding of data modeling, data architecture, and data management principles.
- Proven track record of solving complex problems and delivering high-quality, scalable solutions.
- Excellent communication and collaboration skills to work effectively in a team environment.

[31]

[31] https://careers.fedex.com/data-engineer-iii/job/P25-150097-1?utm_source=CareerSiteOrganic

Data Engineer II

Company: FedEx Dataworks

Category: Professional

Employment Type: Full Time

Worker Sub-Type: Regular Remote Worker

Remote: Yes

Location:

- 3680 Hacks Cross Road, Memphis, TN 38125-8800, United States
- Remote

Knowledge, Skills and Abilities

- Demonstrated expertise working with some of the following common languages and tools:
- Spark (Scala and PySpark), HDFS, Kafka and other high volume data tools
- SQL and NoSQL storage tools, such as MySQL, Postgres, Cassandra, MongoDB and Elastic Search
- Pandas, Scikit-Learn, Matplotlib, TensorFlow, Jupyter and other Python data tools

[32]

Deepak N. · 3rd

Senior Data Engineer | Databricks Certified Data Engineer Associate | IBM Professional Data Engineer | Microsoft Azure Data Engineer Associate| Expert in ETL, Data Warehousing,andReal-TimeAnalytics | PySpark&Apache spark

United States · Contact info

Experience

Sr. Data Engineer

FedEx · Contract

Aug 2022 - Present · 2 yrs 2 mos

Dallas, Texas, United States · Hybrid

- Worked on Spark improving the performance and optimization of the existing algorithms in Hadoop using Spark Context, Spark-SQL, Data Frame, Pair RDD's
- Developed PySpark script to merge static and dynamic files and cleanse the data.
- Created PySpark procedures, functions, packages to load data.
- Solved performance issues in Hive and Pig scripts with understanding of Joins, Group and Aggregation and how does it translate to MapReduce jobs.

[33]

[32] https://careers.fedex.com/data-engineer-ii/job/P25-150088-1?utm_source=CareerSiteOrganic
[33] https://www.linkedin.com/in/deepak-n-7b372915a/

Billy Lin · 3rd
Looking for opportunities as a Software Engineer
San Francisco Bay Area · **Contact info**

Experience

Software Engineer
FedEx
Oct 2020 - Present · 4 yrs
Tennessee, United States

- Experience in NoSQL databases like MongoDB
- Performed Unit Testing using JUnit 5 under TDD environment, used Log4j 2.0 technology for logging and used Maven 3 as build tool for managing dependencies and used Git 2 for version control and Jenkins 2.0 for CI/CD support
- Worked on adding alert rules in SPARK, Hive, Scala, Hadoop.
- Developed unit test using JUnit, Mockito. Implemented Automation test framework using TestNG.
- Hands-on experience in creating, managing, deploying and rolling out applications using Jenkins and Altus(eBay's cloud platform), GitHub and Maven.

Riley Hodge • Virginia Beach, VA
Sr. Big Data/ Scala Developer
FedEx, 2023 - Present

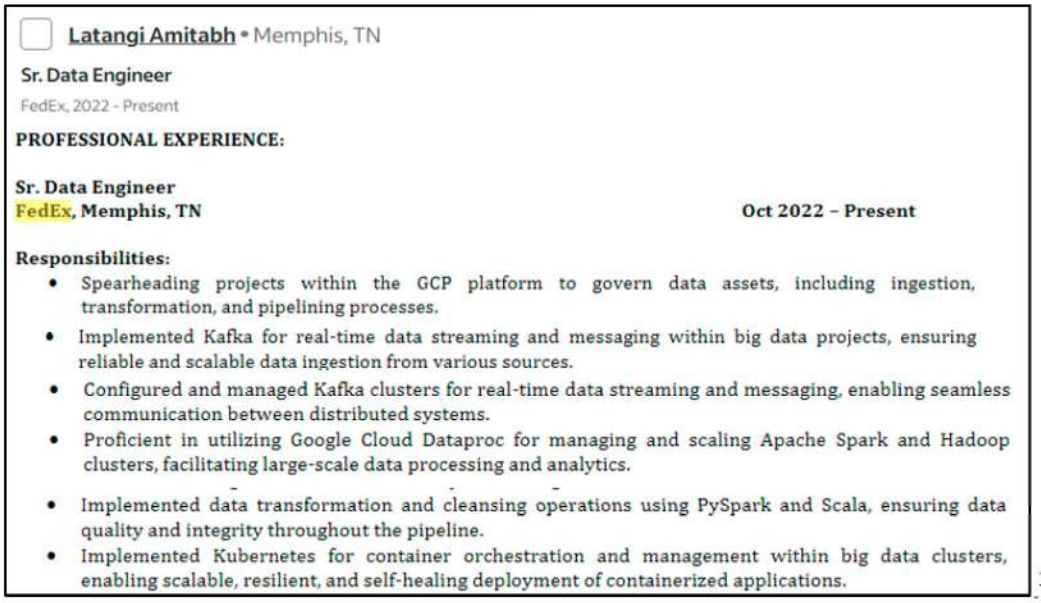
Work Experience

Sr. Big Data/ Scala Developer
FedEx - Memphis, TN
April 2023 to Present

Responsibilities:
· Responsible for building scalable distributed data solutions using Hadoop and migrate legacy Retail applications ETL to Hadoop.
· Wrote the Spark code in Scala to connect to HBASE and read/write data to the HBASE table.
· Extracted data from different databases and to copy into HDFS using Sqoop and have an expertise in using compression techniques to optimize the data storage.
· Implemented Flume, Spark framework for real time data processing. Developed simple to complex Map Reduce jobs using Hive and Pig for analyzing the data.
· Used different Serdes for converting JSON data into pipe separated data.
· Optimized Map/Reduce Jobs to use HDFS efficiently by using various compression mechanisms.
· Built analytics for structured and unstructured data and managing large data ingestion by using Avro, Flume, Thrift, Kafka and Sqoop.
· Worked on scalable distributed computing systems, software architecture, data structures and algorithms using Hadoop, Apache Sparkand Apache Storm etc.
· Ingested streaming data into Hadoop using Spark, Storm Framework and Scala.
· Developed Bankers Rounding UDF for Hive/Pig or Implemented Teradata Rounding in Hive/Pig.
· Continuously monitored and managed the Hadoop Cluster using ClouderaManager.
Environment: Hadoop, Map Reducer, Aws, S3, Redshift, HDFS, Hive, Pig, Spark, Storm, Flume, Thrift, Kafka, Sqoop, Oozie, Impala, SQL, Scala, Java (JDK 1.6), Hadoop (Cloudera) and Eclipse.

[34]

[34] resumes.indeed.com – Smart Search (Login Required).



Latangi Amitabh • Memphis, TN

Sr. Data Engineer

FedEx, 2022 - Present

PROFESSIONAL EXPERIENCE:

Sr. Data Engineer
FedEx, Memphis, TN **Oct 2022 – Present**

Responsibilities:

- Spearheading projects within the GCP platform to govern data assets, including ingestion, transformation, and pipelining processes.
- Implemented Kafka for real-time data streaming and messaging within big data projects, ensuring reliable and scalable data ingestion from various sources.
- Configured and managed Kafka clusters for real-time data streaming and messaging, enabling seamless communication between distributed systems.
- Proficient in utilizing Google Cloud Dataproc for managing and scaling Apache Spark and Hadoop clusters, facilitating large-scale data processing and analytics.
- Implemented data transformation and cleansing operations using PySpark and Scala, ensuring data quality and integrity throughout the pipeline.
- Implemented Kubernetes for container orchestration and management within big data clusters, enabling scalable, resilient, and self-healing deployment of containerized applications.

[35]



Anuradha Modali

Sr. Data Engineer

Patient Pay Inc., 2021 - Present

Data Engineer

FedEx, USA, 2019 - 2021

FedEx, USA **Jul 2019 – Aug 2021**
Data Engineer

Responsibilities:

- Translating customer requirement with HealthCare data (Claims, Membership and Eligibility), FHIR for analysis and implementation.
- Set up and worked on Kerberos authentication principles to establish secure network communication on cluster and testing of HDFS, Hive, Pig and MapReduce to access cluster for new users.
- Build data pipelines in airflow in GCP for ETL related jobs using different airflow operators.
- Experience in GCP Dataproc, GCS, Cloud functions, Big Query
- Developing complex ETL jobs using Jupyter notebooks, PySpark and Python.

[36]

43. For example, Defendants have infringed, and continue to directly infringe, at least claim 1 of the '686 Patent by making, using, offering to sell, selling, and/or importing into the United States products that include HDFS.

44. HDFS performs a method of providing data security for use with a file system. For example, the Apache Hadoop framework provides security features to protect data present in

[35] resumes.indeed.com – Smart Search (Login Required).
[36] resumes.indeed.com – Smart Search (Login Required).

Hadoop clusters. Hadoop uses the HDFS (a file system) to store data. HDFS comprises a name node and multiple data nodes.

> Hadoop, an open-source framework used for distributed storage and processing of big data, provides various security features to protect data and ensure secure access to Hadoop clusters. Here are some of the ways Hadoop is secured:

> 2. Authorization: Hadoop uses Access Control Lists (ACLs) to manage permissions for users and groups accessing data on the Hadoop Distributed File System (HDFS). Additionally, Hadoop provides fine-grained authorization through the use of Apache Ranger, which allows administrators to create policies that define access control for specific resources and actions.[37]

> **NameNode and DataNodes**
>
> HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.[38]

> **The Communication Protocols**
>
> All HDFS communication protocols are layered on top of the TCP/IP protocol. A client establishes a connection to a configurable TCP port on the NameNode machine. It talks the ClientProtocol with the NameNode. The DataNodes talk to the NameNode using the DataNode Protocol. A Remote Procedure Call (RPC) abstraction wraps both the Client Protocol and the DataNode Protocol. By design, the NameNode never initiates any RPCs. Instead, it only responds to RPC requests issued by DataNodes or clients.[39]

45. HDFS performs the step of applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node

[37] https://medium.com/@big_data_landscape/hadoop-security-5dac0ffe209e

[38] http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#NameNode_and_DataNodes

[39] https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

associated with said file. For example, NameNode performs various operations, such as opening and closing of files, and mapping data block locations (e.g., data block number). Mapping (e.g., mapping function) is the process of determining which data block is stored at which location. By way of further example, data blocks are associated with a file. The data block locations are stored in the NameNode's (e.g., index node) memory.

If you take another look at Figure 4-3, you can see the NameNode daemon running on a master node server. All mapping information dealing with the data blocks and their corresponding files is stored in a file named `fsimage`. HDFS is a journaling file system, which means that any data changes are logged in an edit journal that tracks events since the last *checkpoint* — the last time when the edit log was merged with `fsimage`. In HDFS, the edit journal is maintained in a file named `edits` that's stored on the NameNode.

The client opens the file it wishes to read by calling `open()` on the `FileSystem` object, which for HDFS is an instance of `DistributedFileSystem` (step 1 in Figure 3-2). `DistributedFileSystem` calls the namenode, using remote procedure calls (RPCs), to determine the locations of the first few blocks in the file (step 2). For each block, the namenode returns the addresses of the datanodes that have a copy of that block. Furthermore, the datanodes are sorted according to their proximity to the client (according to the topology of the cluster's network; see "Network Topology and Hadoop" on page 70). If the client is itself a datanode (in the case of a MapReduce task, for instance), the client will read from the local datanode if that datanode hosts a copy of the block (see also Figure 2-2 and "Short-circuit local reads" on page 308). [40]

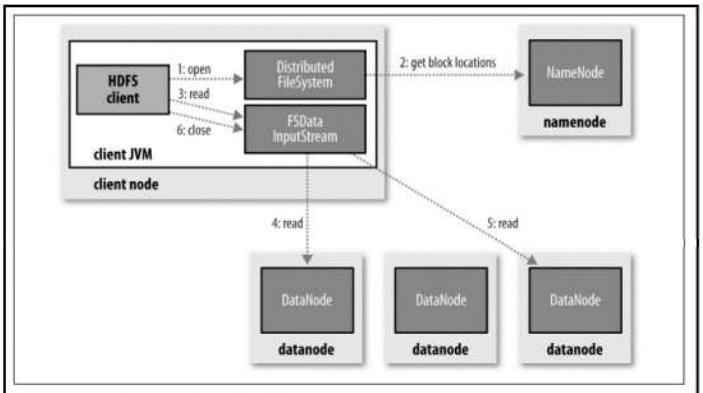


Figure 3-2. A client reading data from HDFS [41]

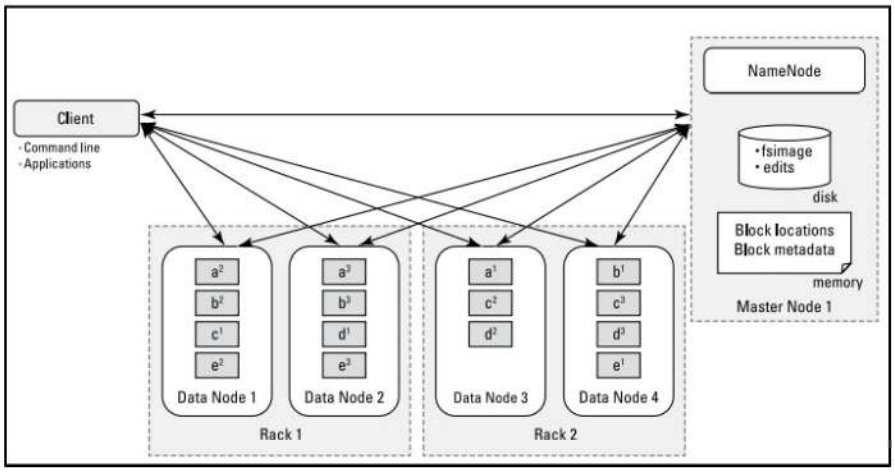
40 https://ia800201.us.archive.org/2/items/HadoopForDummiesDirkDeRoos2014/Hadoop%20For%20Dummies%20-%20%20Dirk%20deRoos%202014.pdf, at Page 74 of 411

41 https://grut-computing.com/HadoopBook.pdf, at Page 97 of 756.

NameNode and DataNodes

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode. [42]

After the startup process is completed, the NameNode has a complete picture of all the data stored in HDFS, and it's ready to receive application requests from Hadoop clients. As data files are added and removed based on client requests, the changes are written to the slave node's disk volumes, journal updates are made to the `edits` file, and the changes are reflected in the block locations and metadata stored in the NameNode's memory (see Figure 4-4).



Throughout the life of the cluster, the DataNode daemons send the NameNode heartbeats (a quick signal) every three seconds, indicating they're active. (This default value is configurable.) Every six hours (again, a configurable default), the DataNodes send the NameNode a block report outlining which file blocks are on their nodes. This way, the NameNode always has a current view of the available resources in the cluster. [43]

[42] https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

[43] https://ia800201.us.archive.org/2/items/HadoopForDummiesDirkDeRoos2014/Hadoop%20For%20Dummies%20-%20%20Dirk%20deRoos%202014.pdf, at Page 75 of 411

46. HDFS performs the step of obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device. For example, NameNode determines "the addresses of the data nodes" (e.g., mapped data block numbers) on which the data block is stored. Data nodes (e.g., storage device(s)) store the data blocks of a file.

> The client opens the file it wishes to read by calling `open()` on the `FileSystem` object, which for HDFS is an instance of `DistributedFileSystem` (step 1 in Figure 3-2). `DistributedFileSystem` calls the namenode, using remote procedure calls (RPCs), to determine the locations of the first few blocks in the file (step 2). For each block, the namenode returns the addresses of the datanodes that have a copy of that block. Furthermore, the datanodes are sorted according to their proximity to the client (according to the topology of the cluster's network; see "Network Topology and Hadoop" on page 70). If the client is itself a datanode (in the case of a MapReduce task, for instance), the client will read from the local datanode if that datanode hosts a copy of the block (see also Figure 2-2 and "Short-circuit local reads" on page 308).

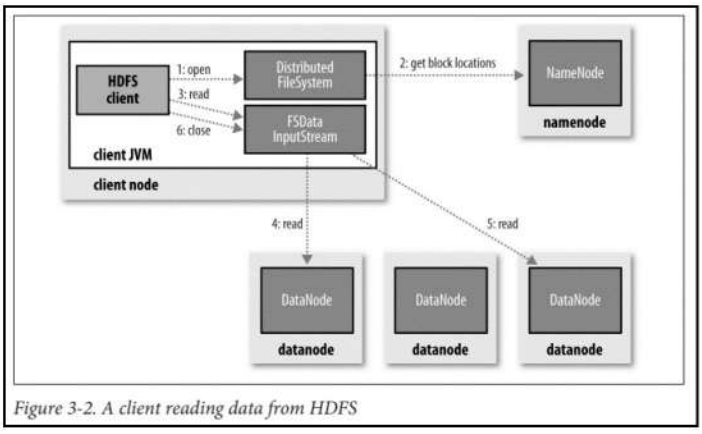


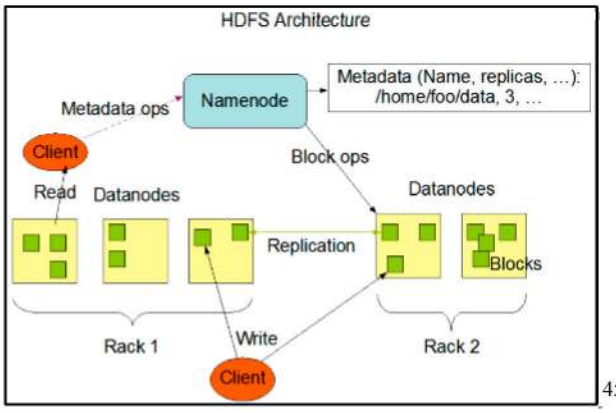
Figure 3-2. A client reading data from HDFS

> One important aspect of this design is that the client contacts datanodes directly to retrieve data and is guided by the namenode to the best datanode for each block. This design allows HDFS to scale to a large number of concurrent clients because the data traffic is spread across all the datanodes in the cluster. Meanwhile, the namenode merely has to service block location requests (which it stores in memory, making them very efficient) and does not, for example, serve data, which would quickly become a bottleneck as the number of clients grew.[44]

[44] https://grut-computing.com/HadoopBook.pdf, at Pages 97-98 of 756.

NameNode and DataNodes

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.



[45]

47. Defendants directly infringe the '686 Patent through their implementation and use of HDFS, performing at least the steps of claim 1.

48. Defendants have indirectly infringed and continue to indirectly infringe one or more claims of the '686 Patent by knowingly and intentionally inducing others, including FedEx customers and end-users, to directly infringe, either literally or under the doctrine of equivalents, by making, using, offering to sell, selling, and/or importing into the United States products that include the infringing technology.

[45] https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

49. Defendants, with knowledge that these products, or the use thereof, infringe the '686 Patent at least as of October 16, 2024,.[46] knowingly and intentionally induced, and continue to knowingly and intentionally induce, direct infringement of the '686 Patent by providing these products to end-users for use in an infringing manner. Additionally, on information and belief, Defendants have adopted a policy of not reviewing the patents of others, including specifically those related to Defendants' specific industry, thereby remaining willfully blind to the '686 Patent at least as early as the issuance of the '686 Patent.

50. Defendants have induced infringement by others, including end-users, with the intent to cause infringing acts by others or, in the alternative, with the belief that there was a high probability that others, including end-users, infringe the '686 Patent, but while remaining willfully blind to the infringement. Defendants have and continue to induce infringement by their customers and end-users by supplying them with instructions on how to operate the infringing technology in an infringing manner, while also making publicly available information on the infringing technology via Defendants' website, product literature and packaging, and other publications..[47]

51. Plaintiffs have suffered damages as a result of Defendants' direct and indirect infringement of the '686 Patent in an amount to be proven at trial.

52. Plaintiffs have suffered, and will continue to suffer, irreparable harm as a result of Defendants' infringement of the '686 Patent, for which there is no adequate remedy at law, unless Defendants' infringement is enjoined by this Court.

---

[46] Plaintiffs sent Defendant FedEx Corporation correspondence on October 8, 2024, providing notice of infringement of the '686 Patent. Therefore, Defendants had knowledge of the '686 Patent as of at least October 2024.

[47] *See, e.g.,* https://www.fedex.com/en-us/developer/get-support.html#:~:text=1.800.GoFedEx%201.800.463.3339%20(,say%20%E2%80%9CInternational%20Services.%E2%80%9D); https://www.fedex.com/en-us/customer-support.html

53. Defendants have committed and continue to commit acts of infringement that Defendants actually knew or should have known constituted an unjustifiably high risk of infringement of at least one valid and enforceable claim of the '686 Patent. Defendants' direct and indirect infringement of the '686 Patent has been and continue to be willful, intentional, deliberate, and/or in conscious disregard of rights under the patent. Plaintiffs are entitled to an award of treble damages, reasonable attorney fees, and costs in bringing this action.

COUNT III
(Infringement of the '332 Patent)

54. Paragraphs 1 through 22 are incorporated by reference as if fully set forth herein.

55. Plaintiffs have not licensed or otherwise authorized Defendants to make, use, offer for sale, sell, or import any products that embody the inventions of the '332 Patent.

56. Defendants have infringed, and continue to directly infringe, the '332 Patent, either literally or under the doctrine of equivalents, without authority and in violation of 35 U.S.C. § 271, by making, using, offering to sell, selling, and/or importing into the United States products that satisfy each and every limitation of one or more claims of the '332 Patent. Such products include Kubernetes.

57. For example, Defendants have infringed, and continue to directly infringe, at least claim 1 of the '332 Patent by making, using, offering to sell, selling, and/or importing into the United States products that include Kubernetes.

58. On information and belief, FedEx has used Kubernetes in the U.S. since at least 2019, as set forth in Paragraph 27.

59. Kubernetes performs the method for hot deployment and/or redeployment in grid computing environment, wherein the grid computing environment includes one or more grid nodes. For example, Kubernetes uses a method of rolling updates to allow for hot deployment and

redeployment (rollouts and rollbacks). Using rolling updates, application containers (e.g., pods) are updated and scheduled on worker nodes of the cluster.

> Kubernetes progressively rolls out changes to your application or its configuration, while monitoring application health to ensure it doesn't kill all your instances at the same time. If something goes wrong, Kubernetes will rollback the change for you. Take advantage of a growing ecosystem of deployment solutions.

[48]

Core Components

A Kubernetes cluster consists of a control plane and one or more worker nodes. Here's a brief overview of the main components:

Control Plane Components

Manage the overall state of the cluster:

kube-apiserver

The core component server that exposes the Kubernetes HTTP API

etcd

Consistent and highly-available key value store for all API server data

kube-scheduler

Looks for Pods not yet bound to a node, and assigns each Pod to a suitable node.

kube-controller-manager

Runs controllers to implement Kubernetes API behavior.

cloud-controller-manager (optional)

Integrates with underlying cloud provider(s).

[48] https://kubernetes.io/

Node Components

Run on every node, maintaining running pods and providing the Kubernetes runtime environment:

kubelet

Ensures that Pods are running, including their containers.

kube-proxy (optional)

Maintains network rules on nodes to implement Services.

Container runtime

Software responsible for running containers. Read Container Runtimes to learn more.[49]

Users expect applications to be available all the time, and developers are expected to deploy new versions of them several times a day. In Kubernetes this is done with rolling updates. A **rolling update** allows a Deployment update to take place with zero downtime. It does this by incrementally replacing the current Pods with new ones. The new Pods are scheduled on Nodes with available resources, and Kubernetes waits for those new Pods to start before removing the old Pods.[50]

60. Kubernetes performs the step of adding a new version of an application release bundle in a repository server. For example, container images are software bundles shared in a registry. To rollout a new version of the software, a container image with the new version is first formed.

Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes **Deployment**. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes control plane schedules the application instances included in that Deployment to run on individual Nodes in the cluster.

[49] https://kubernetes.io/docs/concepts/overview/components/
[50] https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/

When you create a Deployment, you'll need to specify the container image for your application and the number of replicas that you want to run. You can change that information later by updating your Deployment; Module 5 and Module 6 of the bootcamp discuss how you can scale and update your Deployments. [51]

Images

A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.

You typically create a container image of your application and push it to a registry before referring to it in a Pod.

If you don't specify a registry hostname, Kubernetes assumes that you mean the Docker public registry. You can change this behaviour by setting default image registry in container runtime configuration. [52]

61. Kubernetes performs the step of determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server. For example, a control plane has a deployment controller that ensures that the new container image is deployed all across the ReplicaSet of the cluster.

Updating an application

Users expect applications to be available all the time, and developers are expected to deploy new versions of them several times a day. In Kubernetes this is done with rolling updates. A **rolling update** allows a Deployment update to take place with zero downtime. It does this by incrementally replacing the current Pods with new ones. The new Pods are scheduled on Nodes with available resources, and Kubernetes waits for those new Pods to start before removing the old Pods. [53]

[51] https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/
[52] https://kubernetes.io/docs/concepts/containers/images/
[53] https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/

Initially, the pods run the first version of your application—let's suppose its image is tagged as `v1`. You then develop a newer version of the app and push it to an image repository as a new image, tagged as `v2`. You'd next like to replace all the pods with this new version. Because you can't change an existing pod's image after the pod is created, you need to remove the old pods and replace them with new ones running the new image.[54]

Deployments

A Deployment manages a set of Pods to run an application workload, usually one that doesn't maintain state.

A *Deployment* provides declarative updates for Pods and ReplicaSets.

You describe a *desired state* in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

Each time a new Deployment is observed by the Deployment controller, a ReplicaSet is created to bring up the desired Pods. If the Deployment is updated, the existing ReplicaSet that controls Pods whose labels match `.spec.selector` but whose template does not match `.spec.template` are scaled down. Eventually, the new ReplicaSet is scaled to `.spec.replicas` and all old ReplicaSets is scaled to 0.[55]

Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes **Deployment**. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes control plane schedules the application instances included in that Deployment to run on individual Nodes in the cluster.

Once the application instances are created, a Kubernetes Deployment controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster. **This provides a self-healing mechanism to address machine failure or maintenance.**[56]

[54] https://livebook.manning.com/book/kubernetes-in-action/chapter-9/13
[55] https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
[56] https://kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-intro/

62. Kubernetes performs the step of notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use. For example, the Deployment controller notifies Kubelet processes in worker nodes. Each Kubelet process is therefore notified of the desired configuration including the new application and associated protocol information.

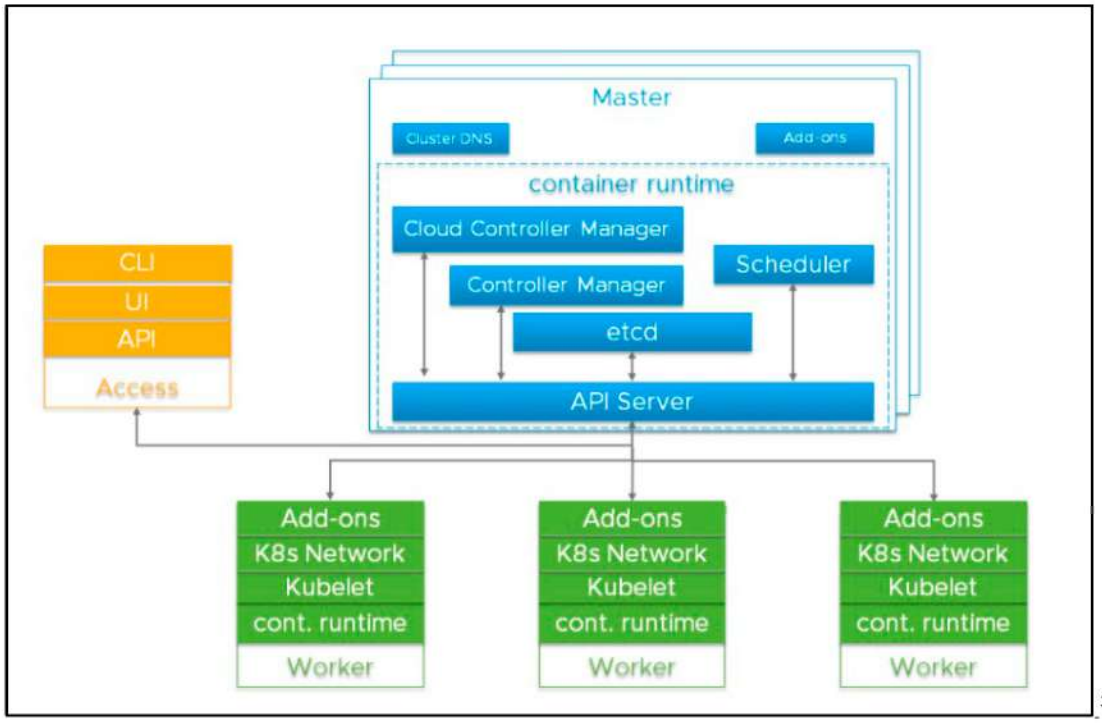
> Each time a new Deployment is observed by the Deployment controller, a ReplicaSet is created to bring up the desired Pods. If the Deployment is updated, the existing ReplicaSet that controls Pods whose labels match `.spec.selector` but whose template does not match `.spec.template` are scaled down. Eventually, the new ReplicaSet is scaled to `.spec.replicas` and all old ReplicaSets is scaled to 0.

[57]

> The kubelet is the primary "node agent" that runs on each node. It can register the node with the apiserver using one of: the hostname; a flag to override the hostname; or specific logic for a cloud provider.
>
> The kubelet works in terms of a PodSpec. A PodSpec is a YAML or JSON object that describes a pod. The kubelet takes a set of PodSpecs that are provided through various mechanisms (primarily through the apiserver) and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

[58]

[57] https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
[58] https://kubernetes.io/docs/concepts/overview/components/#kubelet



[59]

[59] https://kubernetes.io/blog/2018/08/03/out-of-the-clouds-onto-the-ground-how-to-make-kubernetes-production-grade-anywhere/

```
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Thu, 30 Nov 2017 10:56:25 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision=2
Selector:               app=nginx
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
   Containers:
    nginx:
      Image:        nginx:1.16.1
      Port:         80/TCP
      Environment:  <none>
      Mounts:       <none>
    Volumes:        <none>
  Conditions:
    Type           Status  Reason
    ----           ------  ------
    Available      True    MinimumReplicasAvailable
    Progressing    True    NewReplicaSetAvailable
  OldReplicaSets:  <none>
  NewReplicaSet:   nginx-deployment-1564180365 (3/3 replicas created)
  Events:
    Type    Reason             Age   From                   Message
    ----    ------             ----  ----                   -------
    Normal  ScalingReplicaSet  2m    deployment-controller  Scaled up replica set nginx-deployment-2035384211 to 3
    Normal  ScalingReplicaSet  24s   deployment-controller  Scaled up replica set nginx-deployment-1564180365 to 1
    Normal  ScalingReplicaSet  22s   deployment-controller  Scaled down replica set nginx-deployment-2035384211 to 2
    Normal  ScalingReplicaSet  22s   deployment-controller  Scaled up replica set nginx-deployment-1564180365 to 2
    Normal  ScalingReplicaSet  19s   deployment-controller  Scaled down replica set nginx-deployment-2035384211 to 1
    Normal  ScalingReplicaSet  19s   deployment-controller  Scaled up replica set nginx-deployment-1564180365 to 3
    Normal  ScalingReplicaSet  14s   deployment-controller  Scaled down replica set nginx-deployment-2035384211 to 0
```

[60]

63. Kubernetes performs the step of hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers. For example, rolling updates of the new software are made with the rollout command of the kubectl interface. The image is deployed in the worker nodes according to the desired configuration by the Kubelet process.

[60] https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Updating an application

> *Rolling updates allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones.*

Users expect applications to be available all the time, and developers are expected to deploy new versions of them several times a day. In Kubernetes this is done with rolling updates. A **rolling update** allows a Deployment update to take place with zero downtime. It does this by incrementally replacing the current Pods with new ones. The new Pods are scheduled on Nodes with available resources, and Kubernetes waits for those new Pods to start before removing the old Pods.[61]

Deployments

A Deployment manages a set of Pods to run an application workload, usually one that doesn't maintain state.

A *Deployment* provides declarative updates for Pods and ReplicaSets.

You describe a *desired state* in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

- Declare the new state of the Pods by updating the PodTemplateSpec of the Deployment. A new ReplicaSet is created and the Deployment manages moving the Pods from the old ReplicaSet to the new one at a controlled rate. Each new ReplicaSet updates the revision of the Deployment.

[61] https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/

- The `.spec.template` field contains the following sub-fields:
  - The Pods are labeled `app: nginx` using the `.metadata.labels` field.
  - The Pod template's specification, or `.spec` field, indicates that the Pods run one container, `nginx`, which runs the `nginx` Docker Hub image at version 1.14.2.
  - Create one container and name it `nginx` using the `.spec.containers[0].name` field.

[62]

2. To see the rollout status, run:

```
kubectl rollout status deployment/nginx-deployment
```

The output is similar to this:

```
Waiting for rollout to finish: 2 out of 3 new replicas have been updated...
```

or

```
deployment "nginx-deployment" successfully rolled out
```

Get more details on your updated Deployment:

- After the rollout succeeds, you can view the Deployment by running `kubectl get deployments`. The output is similar to this:

```
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           36s
```

[62] https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

The output is similar to this:

```
NAME                               DESIRED   CURRENT   READY   AGE
nginx-deployment-1564180365   3         3         3       6s
nginx-deployment-2035384211   0         0         0       36s
```

[63]

```
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Thu, 30 Nov 2017 10:56:25 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision=2
Selector:               app=nginx
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
   Containers:
    nginx:
      Image:        nginx:1.16.1
      Port:         80/TCP
      Environment:  <none>
      Mounts:       <none>
    Volumes:        <none>
  Conditions:
    Type           Status  Reason
    ----           ------  ------
    Available      True    MinimumReplicasAvailable
    Progressing    True    NewReplicaSetAvailable
  OldReplicaSets:  <none>
  NewReplicaSet:   nginx-deployment-1564180365 (3/3 replicas created)
  Events:
    Type    Reason             Age   From                   Message
    ----    ------             ----  ----                   -------
    Normal  ScalingReplicaSet  2m    deployment-controller  Scaled up replica set nginx-deployment-2035384211 to 3
    Normal  ScalingReplicaSet  24s   deployment-controller  Scaled up replica set nginx-deployment-1564180365 to 1
    Normal  ScalingReplicaSet  22s   deployment-controller  Scaled down replica set nginx-deployment-2035384211 to 2
    Normal  ScalingReplicaSet  22s   deployment-controller  Scaled up replica set nginx-deployment-1564180365 to 2
    Normal  ScalingReplicaSet  19s   deployment-controller  Scaled down replica set nginx-deployment-2035384211 to 1
    Normal  ScalingReplicaSet  19s   deployment-controller  Scaled up replica set nginx-deployment-1564180365 to 3
    Normal  ScalingReplicaSet  14s   deployment-controller  Scaled down replica set nginx-deployment-2035384211 to 0
```

[64]

64. Defendants directly infringe the ’332 Patent through their implementation and use of Kubernetes for rolling updates, performing at least the steps of claim 1.

65. Defendants have indirectly infringed and continue to indirectly infringe one or more claims of the ’332 Patent by knowingly and intentionally inducing others, including FedEx customers and end-users, to directly infringe, either literally or under the doctrine of equivalents,

[63] https://kubernetes.io/docs/concepts/workloads/controllers/deployment/
[64] https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

by making, using, offering to sell, selling, and/or importing into the United States products that include the infringing technology.

66. Defendants, with knowledge that these products, or the use thereof, infringe the '332 Patent at least as of October 16, 2024,.[65] knowingly and intentionally induced, and continue to knowingly and intentionally induce, direct infringement of the '332 Patent by providing these products to end-users for use in an infringing manner. Additionally, on information and belief, Defendants have adopted a policy of not reviewing the patents of others, including specifically those related to Defendants' specific industry, thereby remaining willfully blind to the '332 Patent at least as early as the issuance of the '332 Patent.

67. Defendants have induced infringement by others, including end-users, with the intent to cause infringing acts by others or, in the alternative, with the belief that there was a high probability that others, including end-users, infringe the '332 Patent, but while remaining willfully blind to the infringement. Defendants have and continue to induce infringement by their customers and end-users by supplying them with instructions on how to operate the infringing technology in an infringing manner, while also making publicly available information on the infringing technology via Defendants' website, product literature and packaging, and other publications..[66]

68. Plaintiffs have suffered damages as a result of Defendants' direct and indirect infringement of the '332 Patent in an amount to be proven at trial.

---

[65] Plaintiffs sent Defendant FedEx Corporation correspondence on October 16, 2024, providing notice of infringement of the '332 Patent. Therefore, Defendants had knowledge of the '332 Patent as of at least October 2024.

[66] *See, e.g.,* https://www.fedex.com/en-us/developer/get-support.html#:~:text=1.800.GoFedEx%201.800.463.3339%20(,say%20%E2%80%9CInternational%20Services.%E2%80%9D); https://www.fedex.com/en-us/customer-support.html

69. Plaintiffs have suffered, and will continue to suffer, irreparable harm as a result of Defendants' infringement of the '332 Patent, for which there is no adequate remedy at law, unless Defendants' infringement is enjoined by this Court.

70. Defendants have committed and continue to commit acts of infringement that Defendants knew or should have known constituted an unjustifiably high risk of infringement of at least one valid and enforceable claim of the '332 Patent. Defendants' direct and indirect infringement of the '332 Patent has been and continue to be willful, intentional, deliberate, and/or in conscious disregard of rights under the patent. Plaintiffs are entitled to an award of treble damages, reasonable attorney fees, and costs in bringing this action.

COUNT IV
(Infringement of the '832 Patent)

71. Paragraphs 1 through 22 are incorporated by reference as if fully set forth herein.

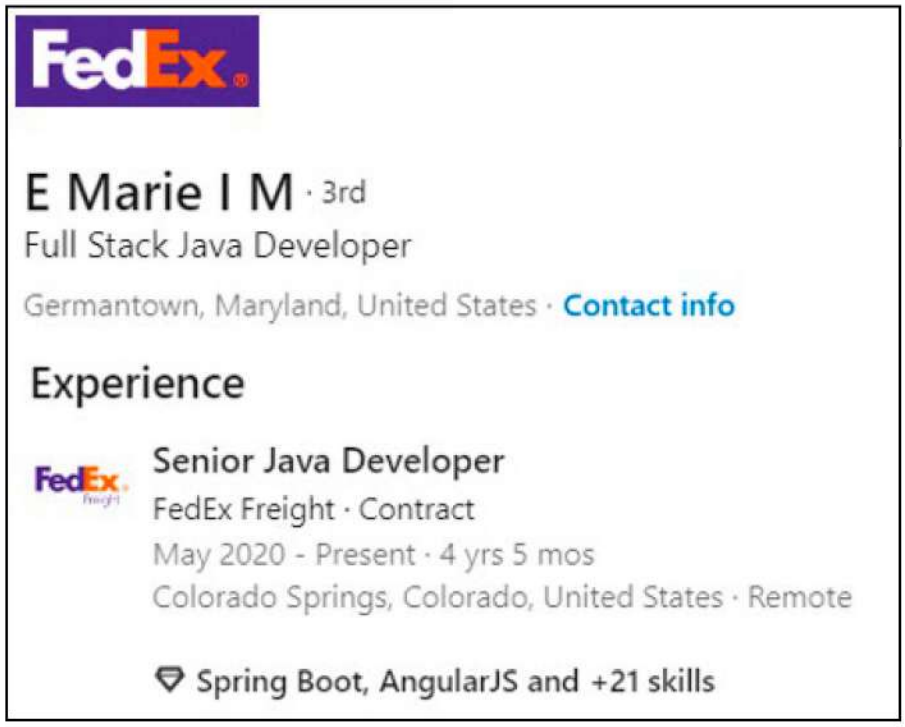
72. Plaintiffs have not licensed or otherwise authorized Defendants to make, use, offer for sale, sell, or import any products that embody the inventions of the '832 Patent.

73. Defendants have directly infringed the '832 Patent, either literally or under the doctrine of equivalents, without authority and in violation of 35 U.S.C. § 271, by making, using, offering to sell, selling, and/or importing into the United States products that satisfy each and every limitation of one or more claims of the '832 Patent. Such products include Apache Kafka Streams using RocksDB Universal Compaction, RocksDB Universal Compaction, Apache Cassandra, and Apache HBase.

74. For example, Defendants have directly infringed at least claim 25 of the '832 Patent by making, using, offering to sell, selling, and/or importing into the United States products that include Apache Kafka Streams using RocksDB Universal Compaction, RocksDB Universal Style Compaction, Apache Cassandra, and Apache HBase.

Apache Kafka Streams using RocksDB Universal Compaction

75. On information and belief, FedEx has been using Apache Kafka Streams using RocksDB Universal Compaction in the U.S. since at least 2020. The default state store type of Apache Kafka Streams is RocksDB. Apache Kafka Streams also uses universal compaction by default for its state stores.





[67]



[68]

[67] https://www.linkedin.com/in/e-marie-i-m-80b41b264/

[68] https://newsroom.fedex.com/newsroom/global/fedex-completes-acquisition-of-shoprunner-expanding-its-e-commerce-capabilities

Software Engineer Intern
ShopRunner
May 2021 - Sep 2021 · 5 mos

•Assist in developing frameworks to productize machine-learning models that give members more product choices.

•Help define and manage big data infrastructure including Kinesis/Kafka streams, Apache Spark and Snowflake data warehouse

•Help evolve service architecture, embracing architecture approaches such as microservices and well-formed APIs to allow architecture to scale both internally and externally [69]

ShopRunner
Full-time · 2 yrs 3 mos
Chicago, Illinois, United States

Senior Principal Engineer
Apr 2021 - Jul 2021 · 4 mos

Architected and lead the rebuild of the product catalog that underpins much of ShopRunner's ecommerce offering. Now replatforming the orders system.
Techs: Kafka, Kotlin, FP, Postgres, Elasticsearch, React

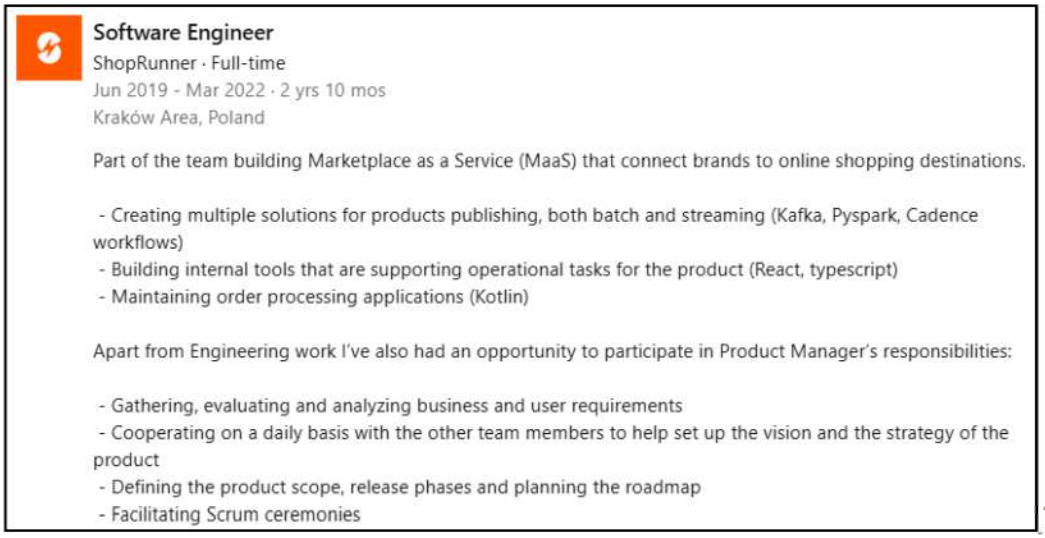
Principal Engineer
May 2019 - Apr 2021 · 2 yrs

Architected and lead the rebuild of the product catalog that underpins much of ShopRunner's ecommerce offering. Now replatforming the orders system.
Techs: Kafka, Kotlin, FP, Postgres, Elasticsearch, React [70]

Software Engineer
ShopRunner · Full-time
Jan 2022 - Dec 2022 · 1 yr
United States

Working on a distributed Kotlin/react application that uses Kafka as a backbone. Millions of messages daily. [71]

[69] https://www.linkedin.com/in/cristian-baeza-01/
[70] https://www.linkedin.com/in/sksamuel/
[71] https://www.linkedin.com/in/corlaez/



Software Engineer
ShopRunner · Full-time
Jun 2019 - Mar 2022 · 2 yrs 10 mos
Kraków Area, Poland

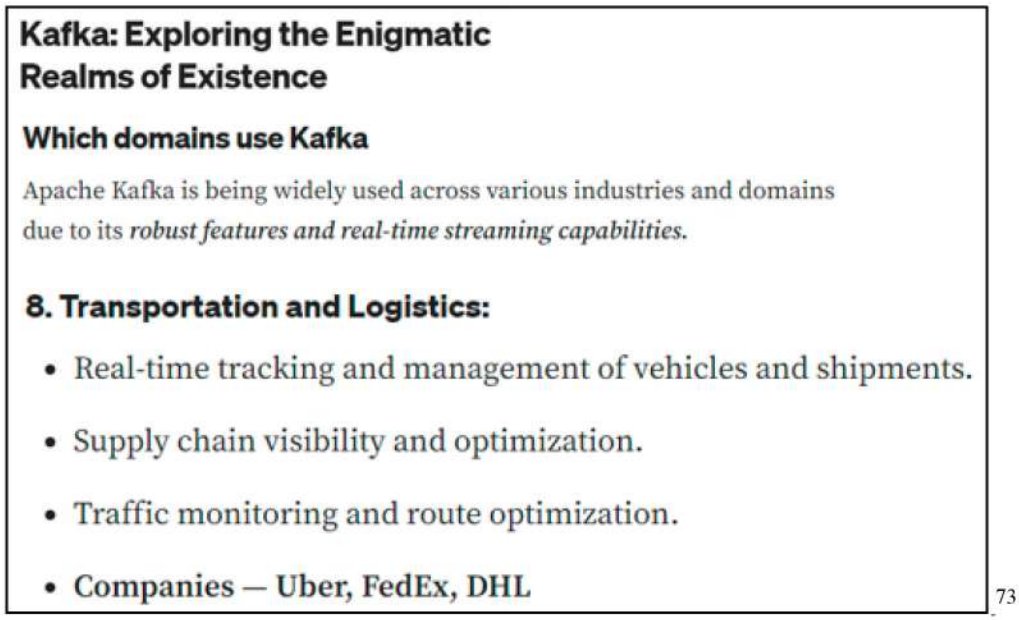
Part of the team building Marketplace as a Service (MaaS) that connect brands to online shopping destinations.

- Creating multiple solutions for products publishing, both batch and streaming (Kafka, Pyspark, Cadence workflows)
- Building internal tools that are supporting operational tasks for the product (React, typescript)
- Maintaining order processing applications (Kotlin)

Apart from Engineering work I've also had an opportunity to participate in Product Manager's responsibilities:

- Gathering, evaluating and analyzing business and user requirements
- Cooperating on a daily basis with the other team members to help set up the vision and the strategy of the product
- Defining the product scope, release phases and planning the roadmap
- Facilitating Scrum ceremonies

[72]



Kafka: Exploring the Enigmatic Realms of Existence

Which domains use Kafka

Apache Kafka is being widely used across various industries and domains due to its *robust features and real-time streaming capabilities.*

8. Transportation and Logistics:

- Real-time tracking and management of vehicles and shipments.
- Supply chain visibility and optimization.
- Traffic monitoring and route optimization.
- **Companies — Uber, FedEx, DHL**

[73]

[72] https://www.linkedin.com/in/renata-gargas/
[73] https://sayantansamanta098.medium.com/kafka-exploring-the-enigmatic-realms-of-existence-a9d102ae2ff

Unlocking Industry Success: The Power of Apache Kafka for Real-time Data Solutions

8. **Transportation and Logistics:**

- **FedEx**: Relies on Kafka for real-time package tracking, route optimization, and supply chain visibility in logistics operations.[74]

FedEx Corporation Technology Stack

Find prospects by the technologies they use. Use this section to learn more about key technologies and tools, tracked by 6sense, used by FedEx Corporation

Apache Kafka

Apache Kafka is an open-source message broker project developed by the Apache Software Fou...[75]

Latangi Amitabh • Memphis, TN

Sr. Data Engineer

FedEx, 2022 - Present

PROFESSIONAL EXPERIENCE:

Sr. Data Engineer
FedEx, Memphis, TN **Oct 2022 – Present**

Responsibilities:

- Spearheading projects within the GCP platform to govern data assets, including ingestion, transformation, and pipelining processes.
- Implemented Kafka for real-time data streaming and messaging within big data projects, ensuring reliable and scalable data ingestion from various sources.
- Configured and managed Kafka clusters for real-time data streaming and messaging, enabling seamless communication between distributed systems.
- Proficient in utilizing Google Cloud Dataproc for managing and scaling Apache Spark and Hadoop clusters, facilitating large-scale data processing and analytics.[76]

[74] https://www.linkedin.com/pulse/mastering-apache-kafka-empowering-industries-data-mohammad-fa-alfard-heonf/
[75] https://6sense.com/company/fedex-corporation/5c3affffd55ae49f1b75e38d
[76] resumes.indeed.com – Smart Search (Login Required).

default.dsl.store

The default state store type used by DSL operators.

Type: string
Default: rocksDB
Valid Values: [rocksDB, in_memory]
Importance: low

[77]

Use level compaction

Kafka Streams uses universal compaction by default for its state stores. However, level compaction has lower space amplification than universal compaction at the cost of lower write rates. The corresponding option that you can tune in your `RocksDBConfigSetter` implementation is `options.setCompactionStyle(CompactionStyle.LEVEL)`.

[78]

76. Apache Kafka Streams using RocksDB Universal Compaction performs a method for monitoring performance of a storage device. For example, Apache Kafka Streams using RocksDB Universal Compaction uses a method of optimizing read and write rates to allow for monitoring the performance of a storage device.

Optimized for Fast Storage

RocksDB is optimized for fast, low latency storage such as flash drives and high-speed disk drives. RocksDB exploits the full potential of high read/write rates offered by flash or RAM.

RocksDB provides several APIs to read KV pairs from a database, including Get and MultiGet for point lookups and Iterator for sequential scanning. These APIs may result in RocksDB reading blocks from SST files on disk storage. The types of blocks and the frequency with which they are read from storage is workload dependent. Some workloads may have a small working set and thus may be able to cache most of the data required, while others may have large working sets and have to read from disk more often. In the latter case, the latency would be much higher and throughput would be lower than the former. They would also be dependent on the characteristics of the underlying storage media, making it difficult to migrate from one medium to another, for example, local flash to disaggregated flash.

[79]

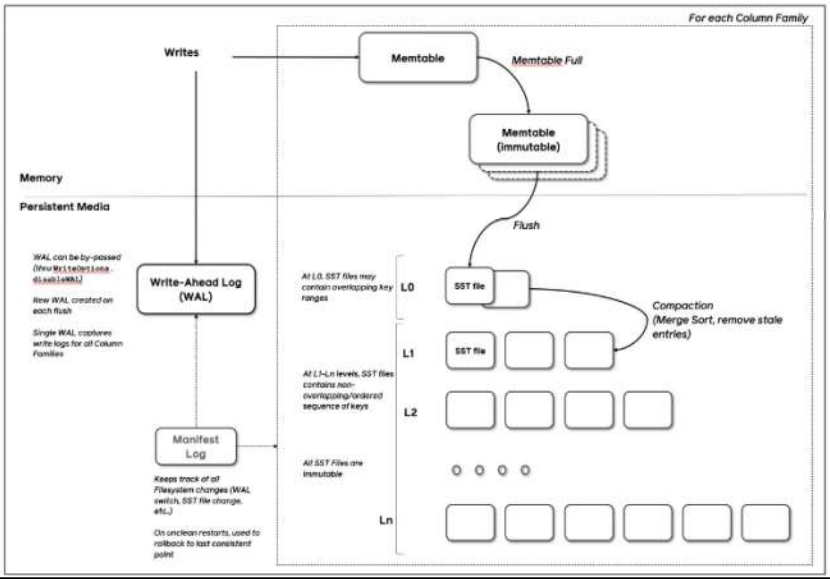
[77] https://kafka.apache.org/documentation/#introduction
[78] https://www.confluent.io/blog/how-to-tune-rocksdb-kafka-streams-state-stores-performance/
[79] https://rocksdb.org/blog/

3. High Level Architecture

RocksDB is a storage engine library of key-value store interface where keys and values are arbitrary byte streams. RocksDB organizes all data in sorted order and the common operations are `Get(key)`, `NewIterator()`, `Put(key, val)`, `Delete(key)`, and `SingleDelete(key)`.

The three basic constructs of RocksDB are **memtable**, **sstfile** and **logfile**. The **memtable** is an in-memory data structure - new writes are inserted into the *memtable* and are optionally written to the **logfile** (aka. Write Ahead Log(WAL)). The logfile is a sequentially-written file on storage. When the memtable fills up, it is flushed to a **sstfile** on storage and the corresponding logfile can be safely deleted. The data in an sstfile is sorted to facilitate easy lookup of keys.

The default format of sstfile is described in more details here.



80

Compaction Styles

Both Level Style Compaction and Universal Style Compaction store data in a fixed number of logical *levels* in the database. More recent data is stored in Level-0 (L0) and older data in higher-numbered levels, up to Lmax. Files in L0 may have overlapping keys, but files in other levels generally form a single sorted run per level.

Level Style Compaction (default) typically optimizes disk footprint vs. logical database size (space amplification) by minimizing the files involved in each compaction step: merging one file in Ln with all its overlapping files in Ln+1 and replacing them with new files in Ln+1.

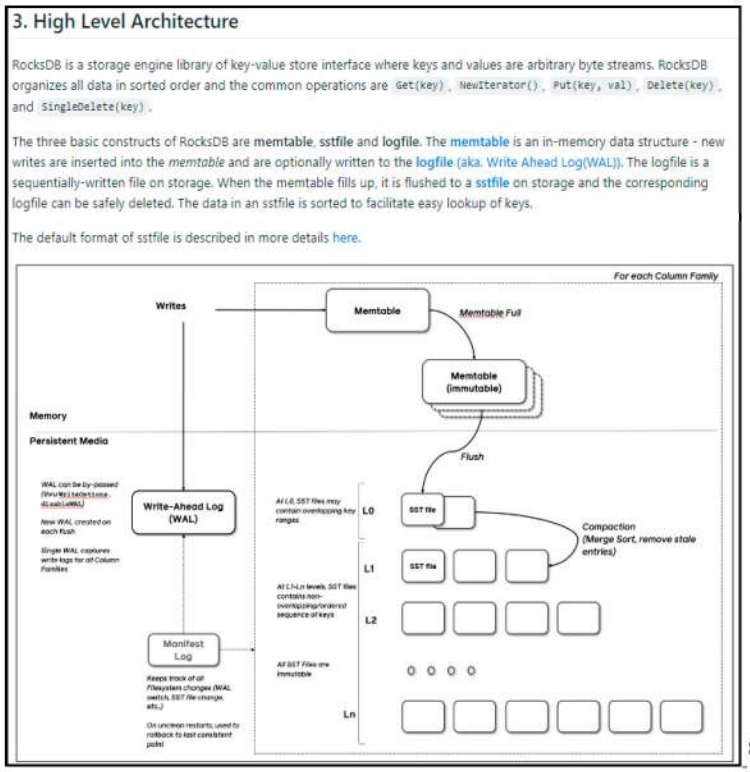
Universal Style Compaction typically optimizes total bytes written to disk vs. logical database size (write amplification) by merging potentially many files and levels at once, requiring more temporary space. Universal typically results in lower write-amplification but higher space- and read-amplification than Level Style Compaction.

81

77. Apache Kafka Streams using RocksDB Universal Compaction performs the step of

[80] https://github.com/facebook/rocksdb/wiki/RocksDB-Overview
[81] https://github.com/facebook/rocksdb/wiki/RocksDB-Overview

intercepting communications between a computer system and said storage device. For example, Apache Kafka Streams using RocksDB Universal Compaction, in response to previously captured performance captured by intercepting communications, inserts new writes into a memtable which are optionally written to the logfile.



82

78. Apache Kafka Streams using RocksDB Universal Compaction performs the step of analyzing said intercepted communications. For example, when the memtable fills up, it is flushed to a sstfile in storage and sorted to facilitate easy lookup of keys. For example, when certain conditions are met, such as the age of data, space amplification, the number of sorted runs while respecting size_ratio, and the number of sorted runs without respecting size_ratio, compaction is triggered and scheduled.

[82] https://github.com/facebook/rocksdb/wiki/RocksDB-Overview

3. High Level Architecture

RocksDB is a storage engine library of key-value store interface where keys and values are arbitrary byte streams. RocksDB organizes all data in sorted order and the common operations are `Get(key)`, `NewIterator()`, `Put(key, val)`, `Delete(key)`, and `SingleDelete(key)`.

The three basic constructs of RocksDB are **memtable**, **sstfile** and **logfile**. The **memtable** is an in-memory data structure - new writes are inserted into the *memtable* and are optionally written to the **logfile** (aka. Write Ahead Log(WAL)). The logfile is a sequentially-written file on storage. When the memtable fills up, it is flushed to a **sstfile** on storage and the corresponding logfile can be safely deleted. The data in an sstfile is sorted to facilitate easy lookup of keys.[83]

Precondition: n >= options.level0_file_num_compaction_trigger

Unless number of sorted runs reaches this threshold, no compaction will be triggered at all.

If the pre-condition is satisfied, there are four conditions. Each of them can trigger a compaction. They are evaluated in order and the selection stops once a compaction has been scheduled. Before going into detail, the overview for picking a compaction is:

1. Try to schedule a compaction by age of data
2. Else if the space amplification constraint has been violated, try to schedule a major compaction
3. Else try to schedule a compaction (minor or major) while respecting size_ratio
4. Else try to schedule a minor compaction without respecting size_ratio[84]

1. Compaction triggered by age of data

For universal style compaction, the aging-based triggering criterion has the highest priority since it is a hard requirement. When trying to pick a compaction, this condition is checked first.

If the condition meets (there are files older than options.periodic_compaction_seconds), then RocksDB proceeds to pick sorted runs for compaction. RocksDB picks sorted runs from oldest to youngest until encountering a sorted run that is being compacted by another compaction. These files will be compacted to bottommost level unless bottommost level is reserved for ingestion behind. In this case, files will be compacted to second bottommost level.

2. Compaction Triggered by Space Amplification

If the estimated *size amplification ratio* is larger than options.compaction_options_universal.max_size_amplification_percent / 100, all files will be compacted to one sorted run.

3. Compaction Triggered by number of sorted runs while respecting size_ratio

If a compaction is not triggered by the space amplification constraint as described above then a compaction can be triggered by the number of sorted runs. This step respects *size ratio trigger* when selecting the sorted runs to compact.

We calculated a value of *size ratio trigger* as

```
size_ratio_trigger = (100 + options.compaction_options_universal.size_ratio) / 100
```

[83] https://github.com/facebook/rocksdb/wiki/Universal-Compaction
[84] https://github.com/facebook/rocksdb/wiki/Terminology

> **4. Compaction Triggered by number of sorted runs without respecting size_ratio**
>
> If none of the previous conditions cause a compaction to be scheduled, then this condition might schedule one. This step does not respect *size ratio trigger* but the logic is otherwise similar to what was described above for condition 3. If we need to compact more than options.compaction_options_universal.max_merge_width number of sorted runs, we cap it to options.compaction_options_universal.max_merge_width.

[85]

79. Apache Kafka Streams using RocksDB Universal Compaction performs the step of reallocating at least some of said data on said storage device to enhance the performance of said storage device based on said analyzed communications. For example, Apache Kafka Streams using RocksDB Universal Compaction will trigger and schedule compactions based on various conditions, which efficiently allocate space.

> In the presence of ongoing writes, compactions are needed for space efficiency, read (query) efficiency, and timely data deletion. Compaction removes key-value bindings that have been deleted or overwritten, and re-organizes data for query efficiency. Compactions may occur in multiple threads if configured.
>
> The entire database is stored in a set of *sstfiles*. When a *memtable* is full, its content is written out to a file in Level-0 (L0) of the LSM tree. RocksDB removes duplicate and overwritten keys in the memtable when it is flushed to a file in L0. In *compaction*, some files are periodically read in and merged to form larger files, often going into the next LSM level (such as L1, up to Lmax).
>
> The overall write throughput of an LSM database directly depends on the speed at which compactions can occur, especially when the data is stored in fast storage like SSD or RAM. RocksDB may be configured to issue concurrent compaction requests from multiple threads. It is observed that sustained write rates may increase by as much as a factor of 10 with multi-threaded compaction when the database is on SSDs, as compared to single-threaded compactions.

[86]

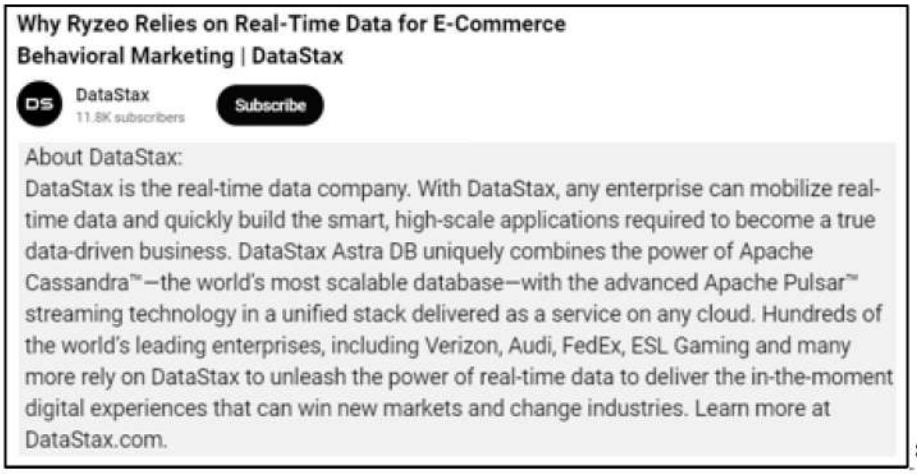
Apache Cassandra

80. On information and belief, FedEx has used Apache Cassandra in the U.S. since at least 2018.

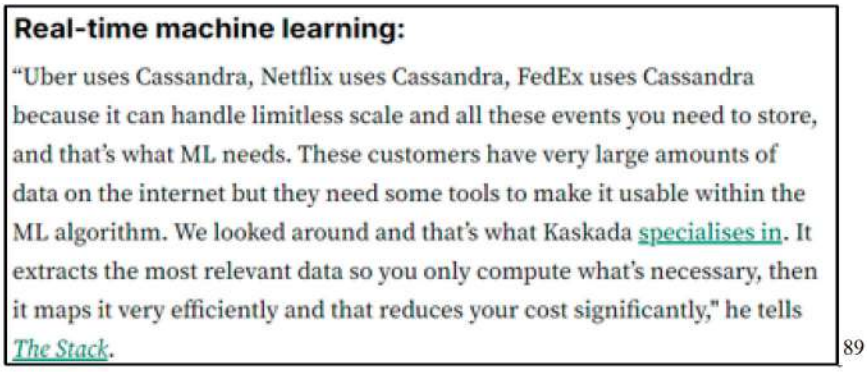
> **Keynote: Data = AI: Leading the Future of Gen AI with Apache Cassandra - Chet Kapoor, DataStax**
>
> The Linux Fou... 189K subscribers | Subscribe | 2 | Share | ...
>
> Join DataStax Chairman & CEO Chet Kapoor to learn:
>
> Why giants like Uber, Netflix, & FedEx already trust Cassandra for the largest AI workloads in the world

[87]

[85] https://github.com/facebook/rocksdb/wiki/Universal-Compaction
[86] https://github.com/facebook/rocksdb/wiki/RocksDB-Overview
[87] https://www.youtube.com/watch?v=flaFAjXXuK8



[88]



[89]



[90]

[88] https://www.youtube.com/watch?v=6B6EuTFAMIg
[89] https://www.thestack.technology/real-time-machine-learning-datastax-kaskada//
[90] https://leadiq.com/c/fedex-corporation/636463d1d4454f2aa3835a80

Shailaja vemula · 3rd
Looking for Full stack developer roles, available to work only on C2C
Orlando, Florida, United States · **Contact info**

Experience

- **Full-stack Developer**
  Contract
  Jul 2020 - Present · 4 yrs 3 mos

Developing the User Interface Screens for presentation using Bootstrap, J Query, AJAX, JSP, HTML, CSS, and Java Script created multi-module Maven project to integrate Spring 5.0, Rest API, Cassandra & micro services and Jenkins for continuous integration.
Moving existing Angular JS controllers and services to Angular7 components for more responsive and less dependency developed RESTAPI's using Spring MVC and spring boot, hosted all Micro services on Pivotal Cloud Foundry (PCF)

[91]

Lucas Oppersdorff • Arlington, TX

Tech Lead/Developer

FedEx, 2021 - Present

Work Experience

Tech Lead/Developer
FedEx - Remote
August 2021 to Present

- (Contract) | Python, Java, SQL, NoSQL, Azure
- Lead engineer on Operations/Planning/Logistics team on last mile delivery project
- Designed and implemented Ingestion, Estimation, Training and Prediction systems
- Implemented ETA and Battery Capacity training and prediction services
- Built Planning service for estimation
- Built Itinerary scheduling service and alt flow Api's
- Azure and Open Source components, AKS, Azure Functions, Azure ServiceBus
- Kubernetes, Jenkins and Azure DevOps CI/CD
- Additional technologies: Cassandra, Redis, RabbitMQ

[92]

[91] https://www.linkedin.com/in/shailaja-vemula-a7770526a/
[92] resumes.indeed.com – Smart Search (Login Required).

Dustin Crain • Newton, NJ

Software Developer 2

FedEx Services, 2022 - Present

Work Experience

Software Developer 2

FedEx Services

March 2022 to Present

As a Software Developer II at FedEx Services, I have led multiple high-impact projects, driving modernization and innovation within the company's tracking systems. My role has encompassed project leadership, architectural design, and end-to-end development, significantly enhancing the efficiency and scalability of the tracking products offered to customers.

- Played a key role in modernizing the existing Spring Boot anonymous tracking API.
- Successfully transitioned from an old Oracle-based solution to a modern, active-anywhere Cassandra database, enabling massive horizontal scalability and handling over 400 *million* queries per day
- Upon promotion to Software Developer 2, led the modernization of the secured tracking API.
- Transitioned from an outdated WebLogic-based solution to a contemporary, scalable Spring Boot solution.
- Achieved significant improvements in scalability and performance, ensuring reliable and efficient tracking services for our customers.
- Led the architecture, development, and deployment of a new webhook tracking system from inception to completion, catering to emerging business needs.
- Spearheaded the creation of a flexible webhook system for business customers to track individual packages, empowering new revenue streams and enhancing tracking capabilities.
- Provided business customers with increased flexibility for tracking individual packages and introduced advanced tracking features, driving business growth and enhancing operational efficiency.
- Languages and Frameworks: Java, Spring Boot, WebLogic (legacy)
- Database: Cassandra, Oracle (legacy)
- Tools: Docker, GitHub Actions, Jenkins
- Development Methodologies: Agile, Scrum [93]

81. Apache Cassandra performs a method for monitoring performance of a storage device. For example, Apache Cassandra monitors performance metrics, such as through "ReadLatency." By way of further example, Apache Cassandra performs compaction, which results in reclaiming disk space.

ReadLatency	Latency	Local read latency for this table.

[94]

[93] resumes.indeed.com – Smart Search (Login Required).
[94] https://cassandra.apache.org/doc/latest/cassandra/operating/metrics.html

Cassandra Compaction is a process of reconciling various copies of data spread across distinct SSTables. Cassandra performs compaction of SSTables as a background activity. Cassandra has to maintain fewer SSTables and fewer copies of each data row due to compactions improving its read performance. Compaction is a crucial area of Cassandra performance and maintenance.

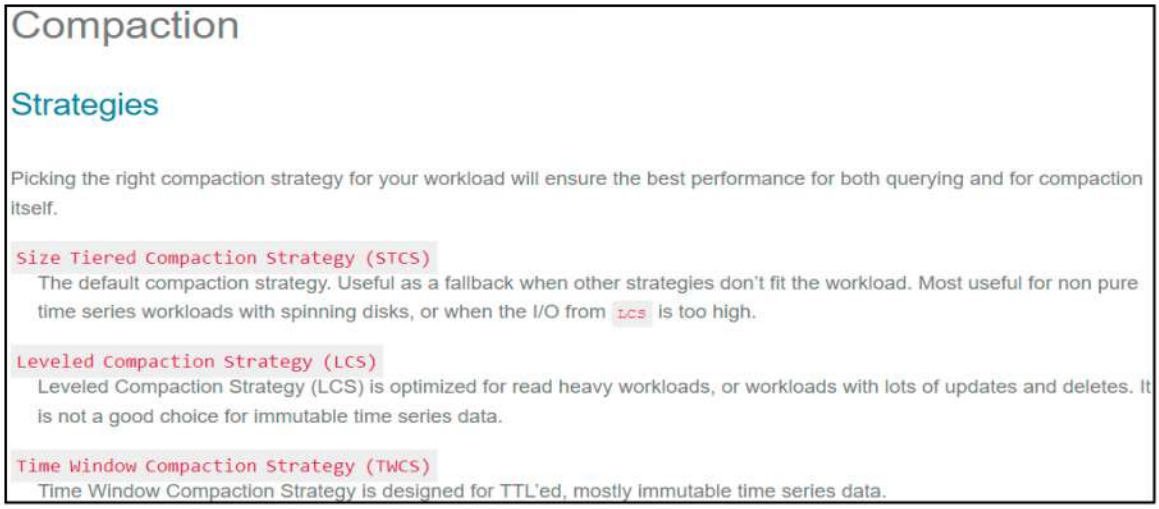
Cassandra's Write Path

To understand the importance of compactions in Cassandra, you must first understand how Cassandra writes data to a disk. The Cassandra write path in a nutshell:

1. Cassandra stores recent writes in memory (in a structure called the Memtable).
2. When enough writes have been made, Cassandra flushes the Memtable to disk. Data on disk is stored in relatively simple data structures called Sorted String Tables (SSTable). At the most simplified level, an SSTable could be described as a sorted array of strings.
3. Before writing a new SSTable, Cassandra merges and pre-sorts the data in the Memtable according to Primary Key. In Cassandra, a Primary Key consists of a Partition Key (the unique key that determines which node the data is stored on) and any Clustering Keys that have been defined.
4. The SSTable is written to disk as a single contiguous write operation. SStables are immutable. Once they are written to disk they are not modified. Any updates to data or deletion of data within an SSTable is written to a new SSTable. If data is updated regularly, Cassandra may need to read from multiple SSTables to retrieve a single row.
5. Compaction operations occur periodically to re-write and combine SSTables. This is required because SSTables are immutable (no modifications once written to disk). Compactions prune deleted data and merge disparate row data into new SSTables in order to reclaim disk space and keep read operations optimized.

If you are unfamiliar with Cassandra's write path, please read **The write path to compaction** from Cassandra Wiki.[95]

[95] https://www.instaclustr.com/blog/apache-cassandra-compaction/



Compaction

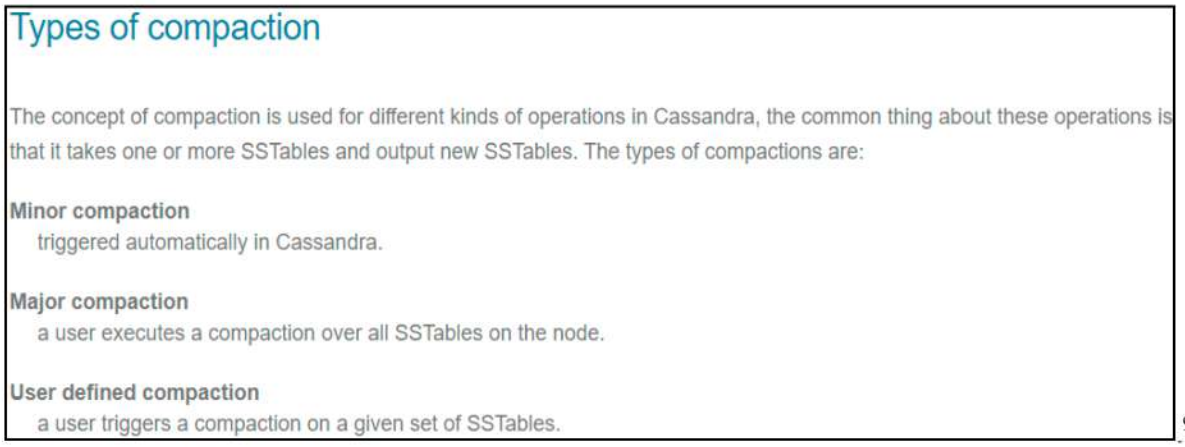
Strategies

Picking the right compaction strategy for your workload will ensure the best performance for both querying and for compaction itself.

`Size Tiered Compaction Strategy (STCS)`
The default compaction strategy. Useful as a fallback when other strategies don't fit the workload. Most useful for non pure time series workloads with spinning disks, or when the I/O from `LCS` is too high.

`Leveled Compaction Strategy (LCS)`
Leveled Compaction Strategy (LCS) is optimized for read heavy workloads, or workloads with lots of updates and deletes. It is not a good choice for immutable time series data.

`Time Window Compaction Strategy (TWCS)`
Time Window Compaction Strategy is designed for TTL'ed, mostly immutable time series data.



Types of compaction

The concept of compaction is used for different kinds of operations in Cassandra, the common thing about these operations is that it takes one or more SSTables and output new SSTables. The types of compactions are:

Minor compaction
triggered automatically in Cassandra.

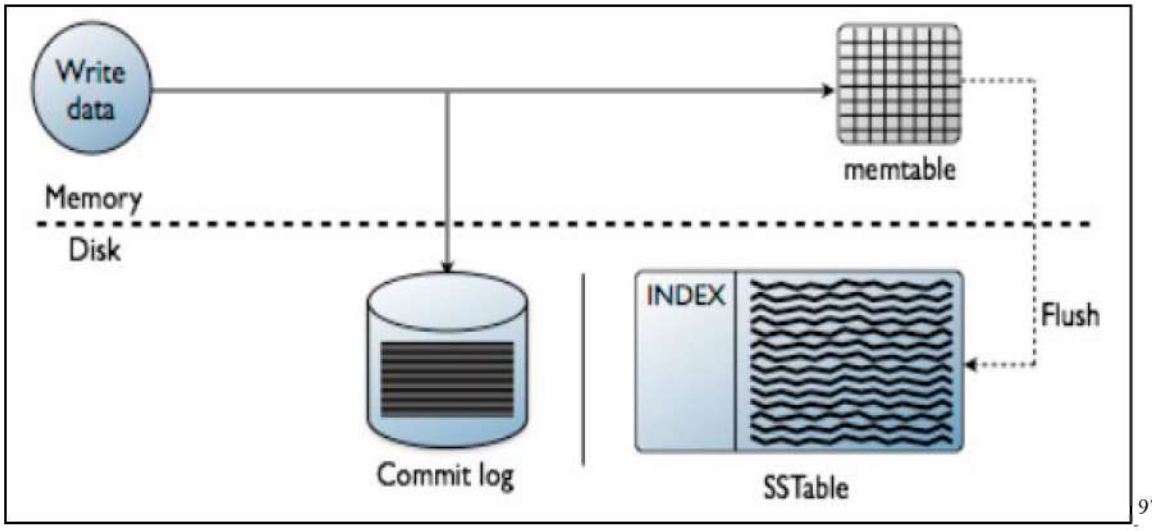
Major compaction
a user executes a compaction over all SSTables on the node.

User defined compaction
a user triggers a compaction on a given set of SSTables.

[96]

82. Apache Cassandra performs the step of intercepting communications between a computer system (e.g., a Cassandra node) and said storage device (e.g., a disk associated with a Cassandra node).

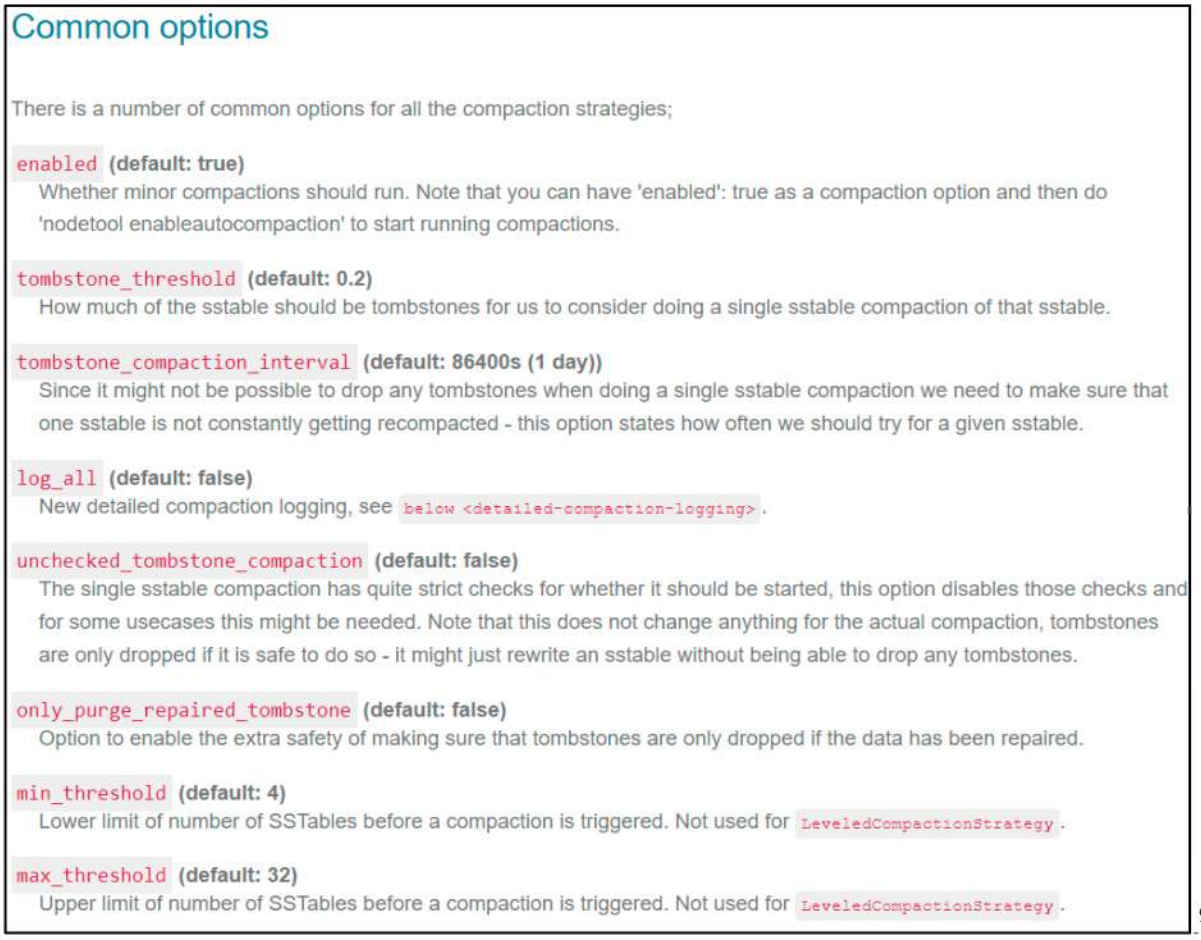
[96] https://cassandra.apache.org/doc/latest/cassandra/operating/compaction/



[97]

83. Apache Cassandra performs the step of analyzing said intercepted communications. For example, the step of analyzing can entail monitoring the memtable and tracking the number of SSTables flushed to a disk and comparing it to the parameters min_threshold and max_threshold to determine whether to perform a minor compaction.

[97] https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/dml/dmlHowDataWritten.html



Common options

There is a number of common options for all the compaction strategies;

`enabled` **(default: true)**
Whether minor compactions should run. Note that you can have 'enabled': true as a compaction option and then do 'nodetool enableautocompaction' to start running compactions.

`tombstone_threshold` **(default: 0.2)**
How much of the sstable should be tombstones for us to consider doing a single sstable compaction of that sstable.

`tombstone_compaction_interval` **(default: 86400s (1 day))**
Since it might not be possible to drop any tombstones when doing a single sstable compaction we need to make sure that one sstable is not constantly getting recompacted - this option states how often we should try for a given sstable.

`log_all` **(default: false)**
New detailed compaction logging, see `below <detailed-compaction-logging>`.

`unchecked_tombstone_compaction` **(default: false)**
The single sstable compaction has quite strict checks for whether it should be started, this option disables those checks and for some usecases this might be needed. Note that this does not change anything for the actual compaction, tombstones are only dropped if it is safe to do so - it might just rewrite an sstable without being able to drop any tombstones.

`only_purge_repaired_tombstone` **(default: false)**
Option to enable the extra safety of making sure that tombstones are only dropped if the data has been repaired.

`min_threshold` **(default: 4)**
Lower limit of number of SSTables before a compaction is triggered. Not used for `LeveledCompactionStrategy`.

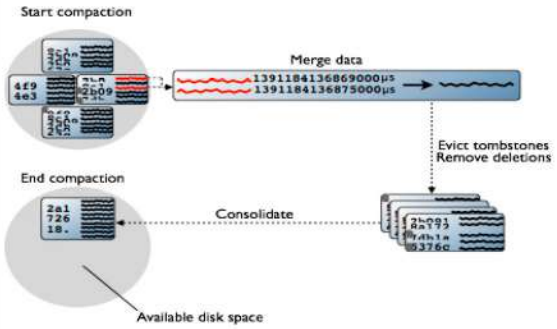
`max_threshold` **(default: 32)**
Upper limit of number of SSTables before a compaction is triggered. Not used for `LeveledCompactionStrategy`.

[98]

84. Apache Cassandra performs the step of reallocating at least some of said data on said storage device to enhance the performance of said storage device based on said analyzed communications. For example, the step of reallocating can occur during minor compaction and entails merging SSTables in response to analysis of previous performance. Continuing with the example, the step of relocating enhances performance of the storage device by: (1) reducing access ("read") time; and (2) making additional disk space available for reuse.

[98] https://cassandra.apache.org/doc/latest/cassandra/operating/compaction/

During compaction, there is a temporary spike in disk space usage and disk I/O because the old and new SSTables co-exist. This diagram depicts the compaction process:



Compaction merges the data in each SSTable by partition key, selecting the latest data for storage based on its timestamp. Cassandra can merge the data performantly, without random IO, because rows are sorted by partition key within each SSTable. After evicting tombstones and removing deleted data, columns, and rows, the compaction process consolidates SSTables into a single file. The old SSTable files are deleted as soon as any pending reads finish using the files. Disk space occupied by old SSTables becomes available for reuse.

Cassandra 2.1 improves read performance after compaction by performing an incremental replacement of compacted SSTables. Instead of waiting for the entire compaction to finish and then throwing away the old SSTable (and cache), Cassandra can read data directly from the new SSTable even before it finishes writing.

As data is written to the new SSTable and reads are directed to it, the corresponding data in the old SSTables is no longer accessed and is evicted from the page cache. Thus begins an incremental process of caching the new SSTable, while directing reads away from the old one. The dramatic cache miss is gone. Cassandra provides predictable high performance even under heavy load.

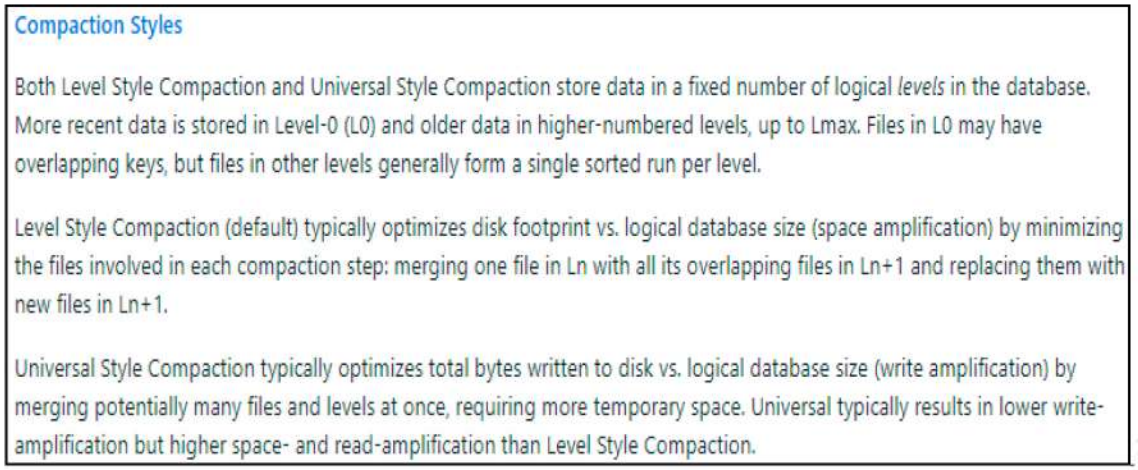
[99]

RocksDB Universal Style Compaction

85. On information and belief, FedEx has used RocksDB Universal Style Compaction in the U.S. since at least 2020, as set forth in Paragraph 75.

86. RocksDB Universal Style Compaction performs the method of monitoring performance of a storage device. For example, RocksDB uses a process called compaction. Two compaction style available with RocksDB are Level Style Compaction and Universal Style Compaction ("USC").

[99] https://docs.datastax.com/en/cassandra-oss/2.1/cassandra/dml/dml_write_path_c.html



Compaction Styles

Both Level Style Compaction and Universal Style Compaction store data in a fixed number of logical *levels* in the database. More recent data is stored in Level-0 (L0) and older data in higher-numbered levels, up to Lmax. Files in L0 may have overlapping keys, but files in other levels generally form a single sorted run per level.

Level Style Compaction (default) typically optimizes disk footprint vs. logical database size (space amplification) by minimizing the files involved in each compaction step: merging one file in Ln with all its overlapping files in Ln+1 and replacing them with new files in Ln+1.

Universal Style Compaction typically optimizes total bytes written to disk vs. logical database size (write amplification) by merging potentially many files and levels at once, requiring more temporary space. Universal typically results in lower write-amplification but higher space- and read-amplification than Level Style Compaction.

[100]

87. RocksDB USC performs the step of intercepting communications between a computer system and said storage device. For example, a storage device can be a disk associated with a RocksDB node. The computer system is a client that functions, e.g., to send write requests that are to be stored on disk. When writing data using RocksDB, the step of intercepting communications between a computer (e.g., a RocksDB node) and a storage device (e.g., a disk associated with the RocksDB node) entails populating data into a memtable.

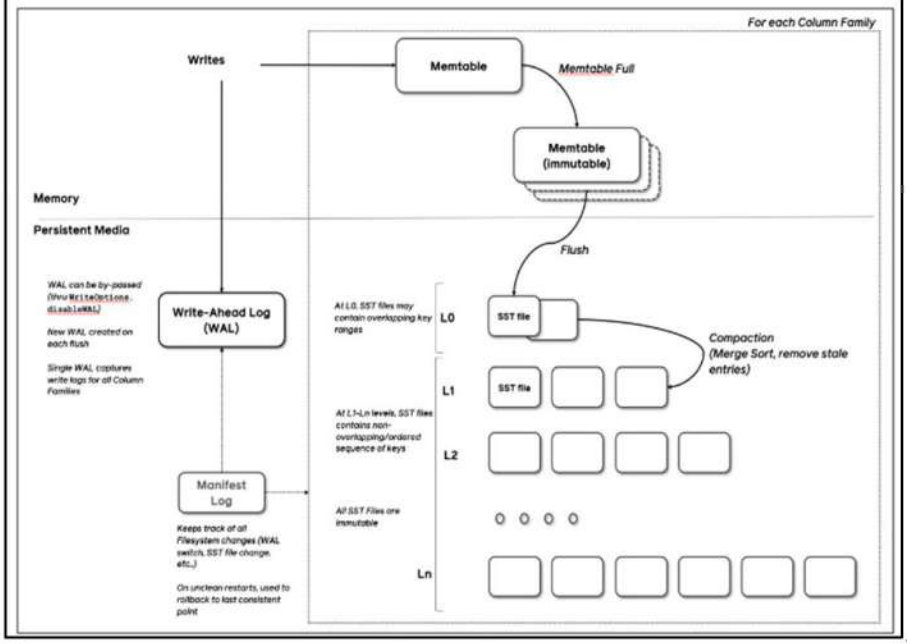
[100] https://github.com/facebook/rocksdb/wiki/RocksDB-Overview

3. High Level Architecture

RocksDB is a storage engine library of key-value store interface where keys and values are arbitrary byte streams. RocksDB organizes all data in sorted order and the common operations are `Get(key)`, `NewIterator()`, `Put(key, val)`, `Delete(key)`, and `SingleDelete(key)`.

The three basic constructs of RocksDB are **memtable**, **sstfile** and **logfile**. The **memtable** is an in-memory data structure - new writes are inserted into the *memtable* and are optionally written to the **logfile** (aka. Write Ahead Log(WAL)). The logfile is a sequentially-written file on storage. When the memtable fills up, it is flushed to a **sstfile** on storage and the corresponding logfile can be safely deleted. The data in an sstfile is sorted to facilitate easy lookup of keys.

The default format of sstfile is described in more details here.



101

88. RocksDB USC performs the step of analyzing said intercepted communications. For example, RocksDB can determine if the memtable size is large enough to be flushed.

3. High Level Architecture

RocksDB is a storage engine library of key-value store interface where keys and values are arbitrary byte streams. RocksDB organizes all data in sorted order and the common operations are `Get(key)`, `NewIterator()`, `Put(key, val)`, `Delete(key)`, and `SingleDelete(key)`.

The three basic constructs of RocksDB are **memtable**, **sstfile** and **logfile**. The **memtable** is an in-memory data structure - new writes are inserted into the *memtable* and are optionally written to the **logfile** (aka. Write Ahead Log(WAL)). The logfile is a sequentially-written file on storage. When the memtable fills up, it is flushed to a **sstfile** on storage and the corresponding logfile can be safely deleted. The data in an sstfile is sorted to facilitate easy lookup of keys.

102

By way of further example, unless the threshold specified by

[101] https://github.com/facebook/rocksdb/wiki/RocksDB-Overview
[102] https://github.com/facebook/rocksdb/wiki/RocksDB-Overview

options.level0_file_num_compaction+trigger, no compaction will be triggered at all.

> **Precondition: n >= options.level0_file_num_compaction_trigger**
>
> Unless number of sorted runs reaches this threshold, no compaction will be triggered at all. [103]

By way of further example, if the precondition is satisfied, then any one of four conditions can trigger a compaction. They are evaluated in order and the selection stops once a compaction has been scheduled. More specifically and as shown in the bottom screenshot, with the precondition satisfied, USC may be triggered by: (1) the age of data; (2) Space Amplification; (3) the number of sorted runs while respecting size_ratio; or (4) number of sorted runs without respecting size_ratio.

> If the pre-condition is satisfied, there are four conditions. Each of them can trigger a compaction. They are evaluated in order and the selection stops once a compaction has been scheduled. Before going into detail, the overview for picking a compaction is:
>
> 1. Try to schedule a compaction by age of data
> 2. Else if the space amplification constraint has been violated, try to schedule a major compaction
> 3. Else try to schedule a compaction (minor or major) while respecting size_ratio
> 4. Else try to schedule a minor compaction without respecting size_ratio [104]

> **1. Compaction triggered by age of data**
>
> For universal style compaction, the aging-based triggering criterion has the highest priority since it is a hard requirement. When trying to pick a compaction, this condition is checked first.
>
> If the condition meets (there are files older than options.periodic_compaction_seconds), then RocksDB proceeds to pick sorted runs for compaction. RocksDB picks sorted runs from oldest to youngest until encountering a sorted run that is being compacted by another compaction. These files will be compacted to bottommost level unless bottommost level is reserved for ingestion behind. In this case, files will be compacted to second bottommost level.

> **2. Compaction Triggered by Space Amplification**
>
> If the estimated *size amplification ratio* is larger than options.compaction_options_universal.max_size_amplification_percent / 100, all files will be compacted to one sorted run.

> **3. Compaction Triggered by number of sorted runs while respecting size_ratio**
>
> If a compaction is not triggered by the space amplification constraint as described above then a compaction can be triggered by the number of sorted runs. This step respects *size ratio trigger* when selecting the sorted runs to compact.
>
> We calculated a value of *size ratio trigger* as
>
> ```
> size_ratio_trigger = (100 + options.compaction_options_universal.size_ratio) / 100
> ```

[103] https://github.com/facebook/rocksdb/wiki/universal-compaction
[104] https://github.com/facebook/rocksdb/wiki/universal-compaction

4. Compaction Triggered by number of sorted runs without respecting size_ratio

If none of the previous conditions cause a compaction to be scheduled, then this condition might schedule one. This step does not respect *size ratio trigger* but the logic is otherwise similar to what was described above for condition 3. If we need to compact more than options.compaction_options_universal.max_merge_width number of sorted runs, we cap it to options.compaction_options_universal.max_merge_width.[105]

89. RocksDB USC performs the step of reallocating at least some of said data on said storage device to enhance the performance of said storage device based on said analyzed communications. For example, reallocation occurs during USCs.

1. Compaction triggered by age of data

For universal style compaction, the aging-based triggering criterion has the highest priority since it is a hard requirement. When trying to pick a compaction, this condition is checked first.

If the condition meets (there are files older than options.periodic_compaction_seconds), then RocksDB proceeds to pick sorted runs for compaction. RocksDB picks sorted runs from oldest to youngest until encountering a sorted run that is being compacted by another compaction. These files will be compacted to bottommost level unless bottommost level is reserved for ingestion behind. In this case, files will be compacted to second bottommost level.

2. Compaction Triggered by Space Amplification

If the estimated *size amplification ratio* is larger than options.compaction_options_universal.max_size_amplification_percent / 100, all files will be compacted to one sorted run.

3. Compaction Triggered by number of sorted runs while respecting size_ratio

If a compaction is not triggered by the space amplification constraint as described above then a compaction can be triggered by the number of sorted runs. This step respects *size ratio trigger* when selecting the sorted runs to compact.

We calculated a value of *size ratio trigger* as

```
size_ratio_trigger = (100 + options.compaction_options_universal.size_ratio) / 100
```

[106]

90. Continuing the example, USCs enhance performance by, among other things, using space more efficiently and increasing write throughput.

4. Compaction Triggered by number of sorted runs without respecting size_ratio

If none of the previous conditions cause a compaction to be scheduled, then this condition might schedule one. This step does not respect *size ratio trigger* but the logic is otherwise similar to what was described above for condition 3. If we need to compact more than options.compaction_options_universal.max_merge_width number of sorted runs, we cap it to options.compaction_options_universal.max_merge_width.[107]

[105] https://github.com/facebook/rocksdb/wiki/Universal-Compaction
[106] https://github.com/facebook/rocksdb/wiki/Universal-Compaction
[107] https://github.com/facebook/rocksdb/wiki/Universal-Compaction

In the presence of ongoing writes, compactions are needed for space efficiency, read (query) efficiency, and timely data deletion. Compaction removes key-value bindings that have been deleted or overwritten, and re-organizes data for query efficiency. Compactions may occur in multiple threads if configured.

The entire database is stored in a set of *sstfiles*. When a *memtable* is full, its content is written out to a file in Level-0 (L0) of the LSM tree. RocksDB removes duplicate and overwritten keys in the memtable when it is flushed to a file in L0. In *compaction*, some files are periodically read in and merged to form larger files, often going into the next LSM level (such as L1, up to Lmax).

The overall write throughput of an LSM database directly depends on the speed at which compactions can occur, especially when the data is stored in fast storage like SSD or RAM. RocksDB may be configured to issue concurrent compaction requests from multiple threads. It is observed that sustained write rates may increase by as much as a factor of 10 with multi-threaded compaction when the database is on SSDs, as compared to single-threaded compactions.

[108]

Apache HBase

91. On information and belief, FedEx has used Apache HBase is the U.S. since at least 2021.

Riley Hodge • Virginia Beach, VA

Sr. Big Data/ Scala Developer

FedEx, 2023 - Present

Work Experience

Sr. Big Data/ Scala Developer

FedEx - Memphis, TN

April 2023 to Present

Responsibilities:

· Responsible for building scalable distributed data solutions using Hadoop and migrate legacy Retail applications ETL to Hadoop.

· Wrote the Spark code in Scala to connect to HBASE and read/write data to the HBASE table.

· Extracted data from different databases and to copy into HDFS using Sqoop and have an expertise in using compression techniques to optimize the data storage.

· Implemented Flume, Spark framework for real time data processing. Developed simple to complex Map Reduce jobs using Hive and Pig for analyzing the data.

· Used different Serdes for converting JSON data into pipe separated data.

· Optimized Map/Reduce Jobs to use HDFS efficiently by using various compression mechanisms.

· Built analytics for structured and unstructured data and managing large data ingestion by using Avro, Flume, Thrift, Kafka and Sqoop.

· Worked on scalable distributed computing systems, software architecture, data structures and algorithms using Hadoop, Apache Sparkand Apache Storm etc.

· Ingested streaming data into Hadoop using Spark, Storm Framework and Scala.

· Developed Bankers Rounding UDF for Hive/Pig or Implemented Teradata Rounding in Hive/Pig.

· Continuously monitored and managed the Hadoop Cluster using ClouderaManager.

Environment: Hadoop, Map Reducer, Aws, S3, Redshift, HDFS, Hive, Pig, Spark, Storm, Flume, Thrift, Kafka, Sqoop, Oozie, Impala, SQL, Scala, Java (JDK 1.6), Hadoop (Cloudera) and Eclipse.

[109]

[108] https://github.com/facebook/rocksdb/wiki/RocksDB-Overview

[109] resumes.indeed.com – Smart Search (Login Required)

FedEx Completes Acquisition of ShopRunner, Expanding its E-Commerce Capabilities

December 28, 2020 07:00 AM
3 min read

[110]

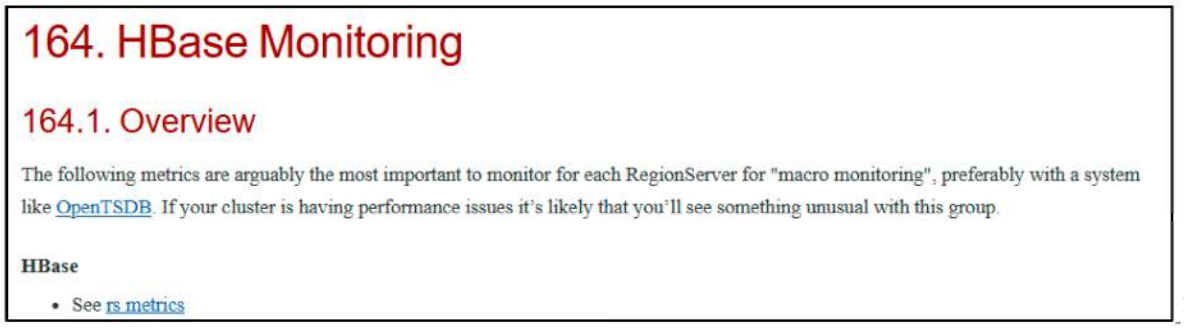
Big Data/Spark Engineer
ShopRunner · Contract
Dec 2018 - Mar 2020 · 1 yr 4 mos
Remote

Devised and documented a novel architecture and development process to shift the existing ETL pipeline into Hadoop-based systems, with a focus on scalability and maintainability.
Leverage Spark PySpark extensively to transform data for creating models to support the Data Science team in their analytics and modeling work.
Constructed Streaming data pipelines with Spark PySpark and HBase, enabling real-time processing of large data streams.
Engineered a reusable Spark-based framework for processing and flattening JSON data that multiple enterprise applications can employ.
Devised a generic framework for preparing data to be fed into machine learning models, ensuring the quality and consistency of the data input.

[111]

[110] https://newsroom.fedex.com/newsroom/global/fedex-completes-acquisition-of-shoprunner-expanding-its-e-commerce-capabilities
[111] https://www.linkedin.com/in/supratik-das-403a1519b/

92. Apache HBase performs the method for monitoring performance of a storage device. For example, the performance of a storage device (e.g., a disk that stores HFiles and/or StoreFiles) is monitored through the use of metrics including the number of read and write requests received.



164. HBase Monitoring

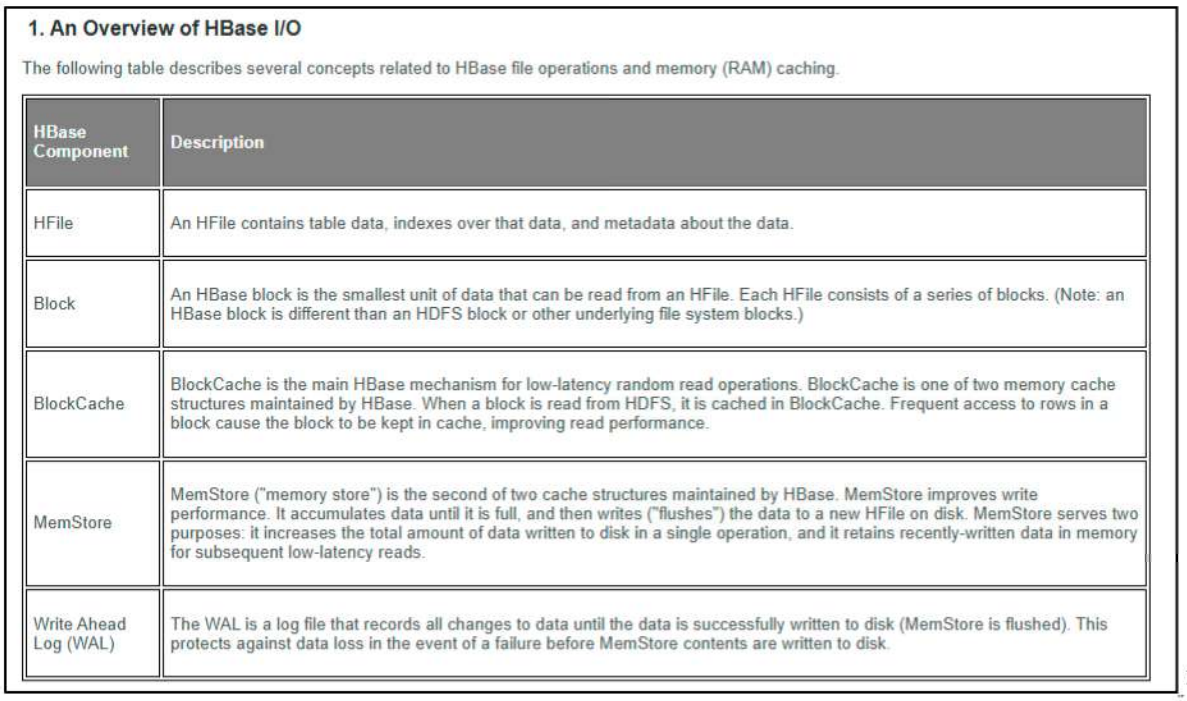
164.1. Overview

The following metrics are arguably the most important to monitor for each RegionServer for "macro monitoring", preferably with a system like OpenTSDB. If your cluster is having performance issues it's likely that you'll see something unusual with this group.

HBase

- See rs metrics

[112]



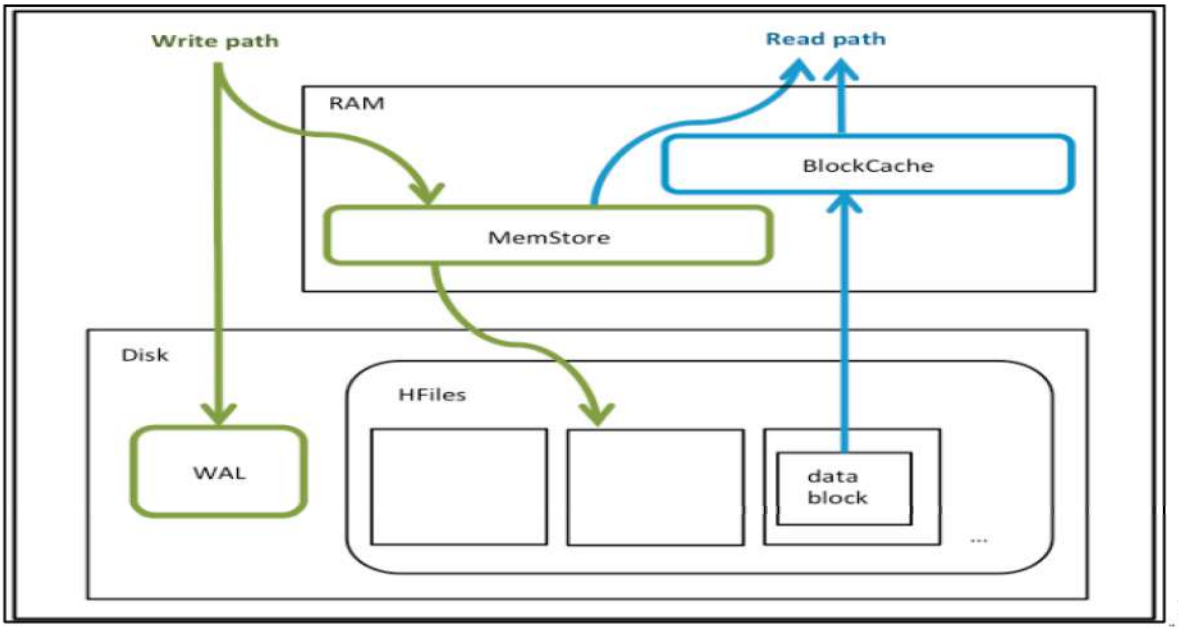
1. An Overview of HBase I/O

The following table describes several concepts related to HBase file operations and memory (RAM) caching.

HBase Component	Description
HFile	An HFile contains table data, indexes over that data, and metadata about the data.
Block	An HBase block is the smallest unit of data that can be read from an HFile. Each HFile consists of a series of blocks. (Note: an HBase block is different than an HDFS block or other underlying file system blocks.)
BlockCache	BlockCache is the main HBase mechanism for low-latency random read operations. BlockCache is one of two memory cache structures maintained by HBase. When a block is read from HDFS, it is cached in BlockCache. Frequent access to rows in a block cause the block to be kept in cache, improving read performance.
MemStore	MemStore ("memory store") is the second of two cache structures maintained by HBase. MemStore improves write performance. It accumulates data until it is full, and then writes ("flushes") the data to a new HFile on disk. MemStore serves two purposes: it increases the total amount of data written to disk in a single operation, and it retains recently-written data in memory for subsequent low-latency reads.
Write Ahead Log (WAL)	The WAL is a log file that records all changes to data until the data is successfully written to disk (MemStore is flushed). This protects against data loss in the event of a failure before MemStore contents are written to disk.

[113]

[112] https://hbase.apache.org/book.html#ops.monitoring
[113] https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.2.0/bk_Sys_Admin_Guides/content/sysadmin_HBase_io_overview.html



114

93. Apache HBase performs the step of intercepting communications between a computer system and said storage device. For example, a computer system is a client that functions, e.g., to send write requests, that are to be stored on a disk. When writing data, the communications between a computer system (e.g., a client) and a storage device (e.g., a disk), are intercepted by a RegionServer populating data into a MemStore prior to it being written to disk.

> BlockCache and MemStore reside in random-access memory (RAM); HFiles and the Write Ahead Log are persisted to HDFS.
>
> Figure 1 shows write and read paths (simplified):
>
> - During write operations, HBase writes to WAL and MemStore. Data is flushed from MemStore to disk according to size limits and flush interval.
> - During read operations, HBase reads the block from BlockCache or MemStore if it is available in those caches. Otherwise it reads from disk and stores a copy in BlockCache.

115

[114] https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.2.0/bk_Sys_Admin_Guides/content/sysadmin_HBase_io_overview.html
[115] https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.2.0/bk_Sys_Admin_Guides/content/sysadmin_HBase_io_overview.html

Figure 1 shows write and read paths (simplified):

- During write operations, HBase writes to WAL and MemStore. Data is flushed from MemStore to disk according to size limits and flush interval.
- During read operations, HBase reads the block from BlockCache or MemStore if it is available in those caches. Otherwise it reads from disk and stores a copy in BlockCache.

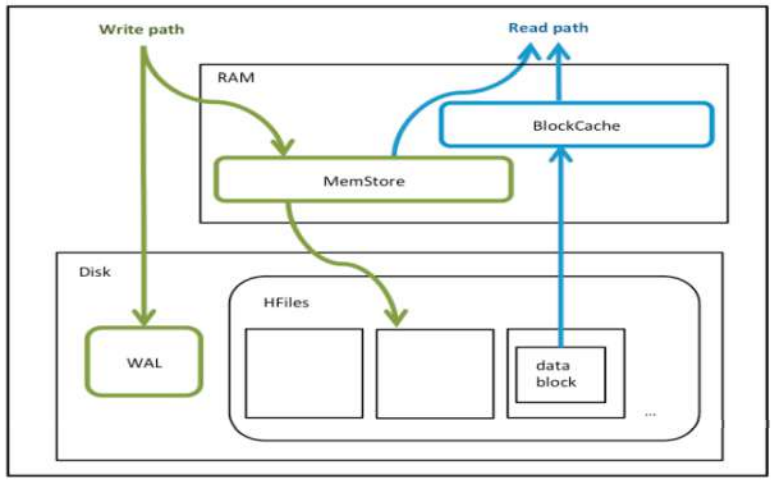


Figure 1. HBase read and write operations

116

94. Apache HBase performs the step of analyzing said intercepted communications. For example, Apache HBase analyzes the intercepted communications by: (1) monitoring the MemStore size; and (2) tracking the number of StoreFiles flushed to disk and comparing it to the value of the parameter hbase.hstore.compaction.min to determine whether to perform a compaction.

`hbase.regionserver.global.memstore.size`

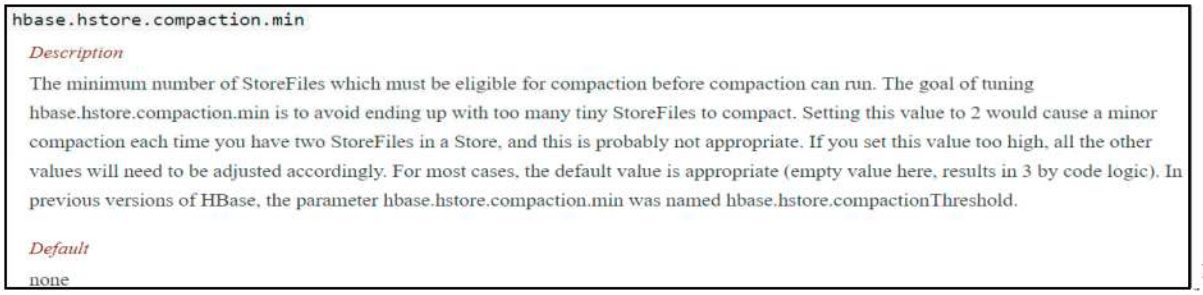
Description

Maximum size of all memstores in a region server before new updates are blocked and flushes are forced. Defaults to 40% of heap (0.4). Updates are blocked and flushes are forced until size of all memstores in a region server hits hbase.regionserver.global.memstore.size.lower.limit. The default value in this configuration has been intentionally left empty in order to honor the old hbase.regionserver.global.memstore.upperLimit property if present.

Default

none

[116] https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.2.0/bk_Sys_Admin_Guides/content/sysadmin_HBase_io_overview.html



`hbase.hstore.compaction.min`

Description

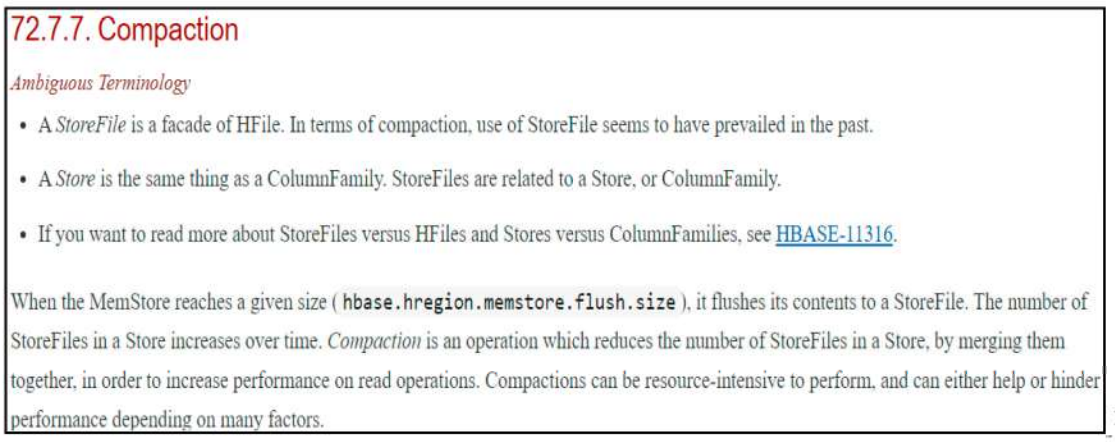
The minimum number of StoreFiles which must be eligible for compaction before compaction can run. The goal of tuning hbase.hstore.compaction.min is to avoid ending up with too many tiny StoreFiles to compact. Setting this value to 2 would cause a minor compaction each time you have two StoreFiles in a Store, and this is probably not appropriate. If you set this value too high, all the other values will need to be adjusted accordingly. For most cases, the default value is appropriate (empty value here, results in 3 by code logic). In previous versions of HBase, the parameter hbase.hstore.compaction.min was named hbase.hstore.compactionThreshold.

Default

none

[117]

95. Apache HBase performs the step of reallocating at least some of said data on said storage device to enhance the performance of said storage device based on said analyzed communications. For example, compaction reallocates at least some of the data (e.g., StoreFiles) on the storage device (e.g., disk) to enhance performance of the storage device by, for example, increasing performance on read operations. The reallocating is based upon the analyzed communications including use of various parameters such as hbase.regionserver.global.memstore.size and hbase.hstore.compaction.min.



72.7.7. Compaction

Ambiguous Terminology

- A *StoreFile* is a facade of HFile. In terms of compaction, use of StoreFile seems to have prevailed in the past.
- A *Store* is the same thing as a ColumnFamily. StoreFiles are related to a Store, or ColumnFamily.
- If you want to read more about StoreFiles versus HFiles and Stores versus ColumnFamilies, see HBASE-11316.

When the MemStore reaches a given size (`hbase.hregion.memstore.flush.size`), it flushes its contents to a StoreFile. The number of StoreFiles in a Store increases over time. *Compaction* is an operation which reduces the number of StoreFiles in a Store, by merging them together, in order to increase performance on read operations. Compactions can be resource-intensive to perform, and can either help or hinder performance depending on many factors.

[118]

96. Defendants have directly infringed the '832 Patent through their use, promotional demonstrations, testing, repairs, and instructional guidance of Apache Kafka Streams using RocksDB Universal Style Compaction, RocksDB Universal Style Compaction, Apache Cassandra, and Apache HBase, performing at least the steps of claim 25.

[117] https://hbase.apache.org/book.html
[118] https://hbase.apache.org/2.2/book.html

97. Plaintiffs have suffered damages as a result of Defendants' direct infringement of the '832 Patent in an amount to be proven at trial.

DEMAND FOR JURY TRIAL

Plaintiffs hereby demand a jury for all issues so triable.

PRAYER FOR RELIEF

WHEREFORE, Plaintiffs pray for relief against Defendants as follows:

a. Entry of judgment declaring that Defendants have directly and/or indirectly infringed one or more claims of each of the Patents-in-Suit;

b. Entry of judgment declaring that Defendants' infringement of the '416 Patent, the '686 Patent, and the '332 Patent is willful;

c. An order pursuant to 35 U.S.C. § 283 permanently enjoining Defendants, their officers, agents, servants, employees, attorneys, and those persons in active concert or participation with them, from further acts of infringement of the '416 Patent, the '686 Patent, and the'332 Patent;

d. An order awarding damages sufficient to compensate Plaintiffs for Defendants' infringement of the Patents-in-Suit, but in no event less than a reasonable royalty, together with interest and costs;

e. Entry of judgment declaring that this case is exceptional and awarding Plaintiffs their costs and reasonable attorney fees under 35 U.S.C. § 285; and

f. Such other and further relief as the Court deems just and proper.

Dated: May 9, 2025

Respectfully submitted,

/s/ Vincent J. Rubino, III
Alfred R. Fabricant
NY Bar No. 2219392
Email: ffabricant@fabricantllp.com
Peter Lambrianakos
NY Bar No. 2894392
Email: plambrianakos@fabricantllp.com
Vincent J. Rubino, III
NY Bar No. 4557435
Email: vrubino@fabricantllp.com
FABRICANT LLP
411 Theodore Fremd Avenue
Suite 206 South
Rye, New York 10580
Telephone: (212) 257-5797
Facsimile: (212) 257-5796

ATTORNEYS FOR PLAINTIFFS VALTRUS INNOVATIONS LTD. and KEY PATENT INNOVATIONS LTD.

EXHIBIT A

US008370416B2

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

(10) **Patent No.: US 8,370,416 B2**
(45) **Date of Patent: Feb. 5, 2013**

(54) **COMPATIBILITY ENFORCEMENT IN CLUSTERED COMPUTING SYSTEMS**

(75) Inventors: **David J. Hoover**, Roseville, CA (US); **Bernard A. Reed, III**, Colorado Springs, CO (US); **Marjorie Krueger**, Roseville, CA (US); **Douglas L. Voigt**, Boise, ID (US)

(73) Assignee: **Hewlett-Packard Development Company, L.P.**, Houston, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1351 days.

(21) Appl. No.: **11/411,732**

(22) Filed: **Apr. 26, 2006**

(65) **Prior Publication Data**

US 2007/0255813 A1 Nov. 1, 2007

(51) **Int. Cl.**
G06F 15/16 (2006.01)

(52) **U.S. Cl.** **709/201**; 709/203; 709/226; 707/610; 707/694; 718/100

(58) **Field of Classification Search** 709/201, 709/203, 226; 717/177; 707/610, 694, 827; 718/100

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,785,713	B1 *	8/2004	Freeman et al.	709/208
6,788,648	B1	9/2004	Peterson	
6,959,320	B2 *	10/2005	Shah et al.	709/203
7,035,918	B1 *	4/2006	Redding et al.	709/223
7,711,647	B2 *	5/2010	Gunaseelan et al.	705/59
8,010,513	B2 *	8/2011	Zhang et al.	707/694
8,069,119	B2 *	11/2011	Curren	705/59
2001/0049717	A1 *	12/2001	Freeman et al.	709/203
2003/0126202	A1 *	7/2003	Watt	709/203
2003/0144854	A1	7/2003	Macholda	
2003/0187983	A1	10/2003	Rajic	
2003/0233439	A1	12/2003	Stone	
2004/0193861	A1	9/2004	Michaelis	
2005/0022173	A1	1/2005	Kanade	
2005/0234846	A1 *	10/2005	Davidson et al.	707/1
2006/0036894	A1 *	2/2006	Bauer et al.	714/4
2006/0106728	A1 *	5/2006	Yellai et al.	705/59
2006/0282519	A1	12/2006	Trevathan	
2009/0276771	A1 *	11/2009	Nickolov et al.	717/177
2011/0145328	A1 *	6/2011	Wang et al.	709/203
2011/0153697	A1 *	6/2011	Nickolov et al.	707/827

(Continued)

FOREIGN PATENT DOCUMENTS

GB	2362969	12/2001
JP	H07-141305 A	6/1995

(Continued)

OTHER PUBLICATIONS

UK Search Report.

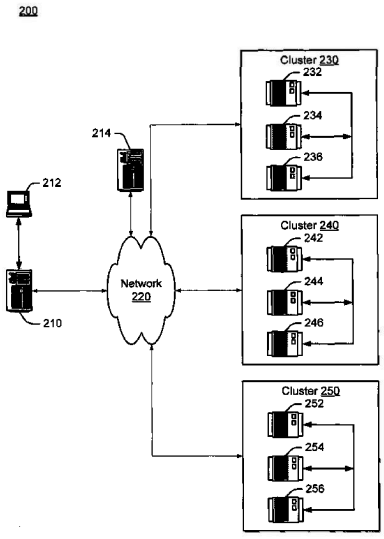
(Continued)

Primary Examiner — Thu Ha Nguyen

(57) **ABSTRACT**

In one embodiment a method of creating a clustered computing system, comprises storing license information for a computing cluster in a memory module associated with a computing node, wherein the licensing information includes a bundle-type parameter that identifies a characteristic of the cluster and one or more node license parameters that identifies a characteristic of a computing node, initializing the computing cluster in a first computing node, adding one or more available computing nodes to the computing cluster; and activating the computing cluster when the computing cluster includes a number of nodes that complies with the bundle-type parameter.

18 Claims, 6 Drawing Sheets



U.S. PATENT DOCUMENTS

2011/0179112	A1*	7/2011	Ravichandran et al.	709/203
2011/0251992	A1*	10/2011	Bethlehem et al.	707/610
2011/0276699	A1*	11/2011	Pedersen	709/227
2011/0307548	A1*	12/2011	Fisk et al.	709/203
2012/0011514	A1*	1/2012	Burton et al.	718/100

FOREIGN PATENT DOCUMENTS

JP	H09-171502	A	6/1997
JP	2003-323224	A	11/2003
JP	2004-215064	A	7/2004
JP	2004-234559	A	8/2004
JP	2005-235019	A	9/2005
WO	WO9939269		8/1999

OTHER PUBLICATIONS

Japanese Notice of Rejection in corresponding Japanese patent application No. 2007-115272, dated Mar. 12, 2010, English translation.
English translation of Japanese Notice of Rejection in corresponding Japanese Patent Application No. 2007-115272, dated Jul. 5, 2010.

* cited by examiner

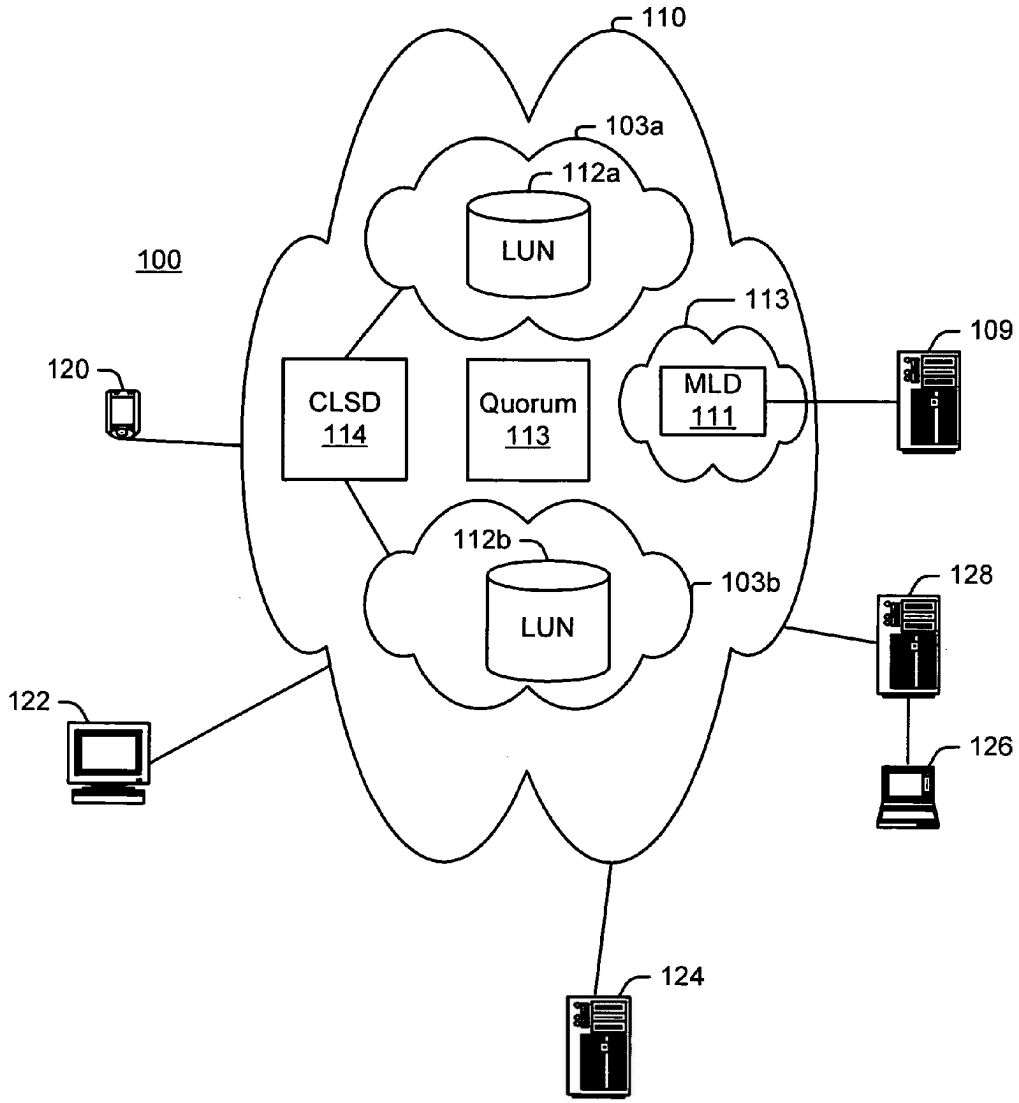

110
103a
112a
LUN
100
113
109
120
CLSD
114
Quorum
113
MLD
111
112b
103b
128
LUN
122
126
124

Fig. 1

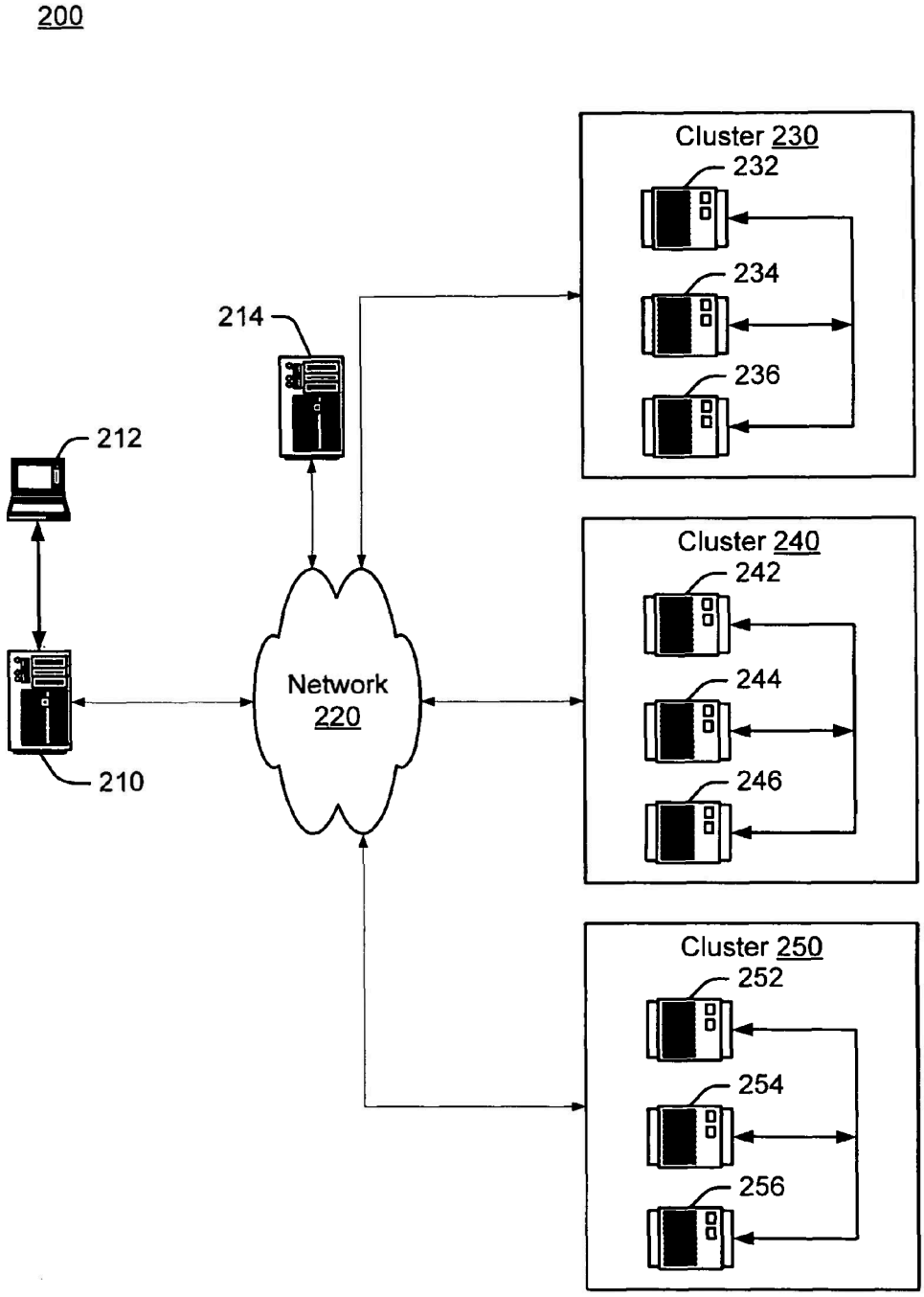

200
Cluster 230
232
234
236
214
212
Cluster 240
242
244
246
Network
220
210
Cluster 250
252
254
256

Fig. 2

Case 2:25-cv-00516 Document 1-1 Filed 05/09/25 Page 6 of 15 PageID #: 80

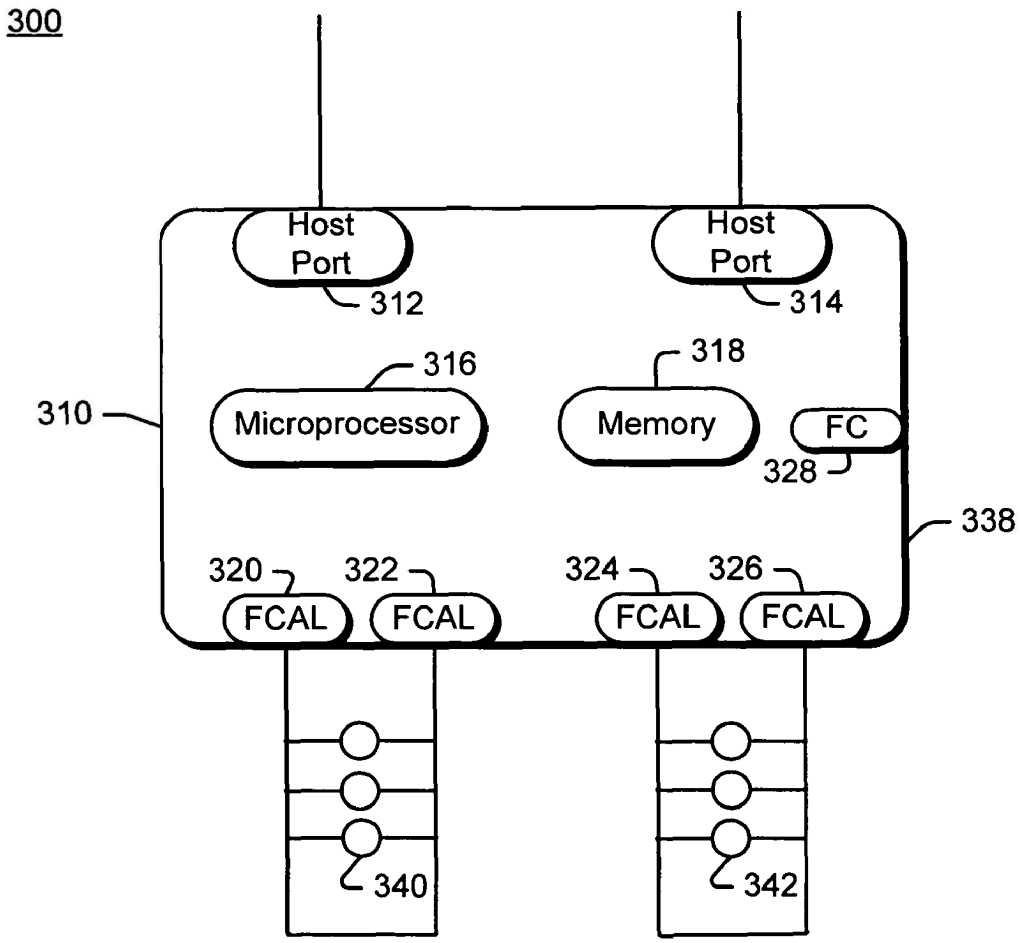


Fig. 3

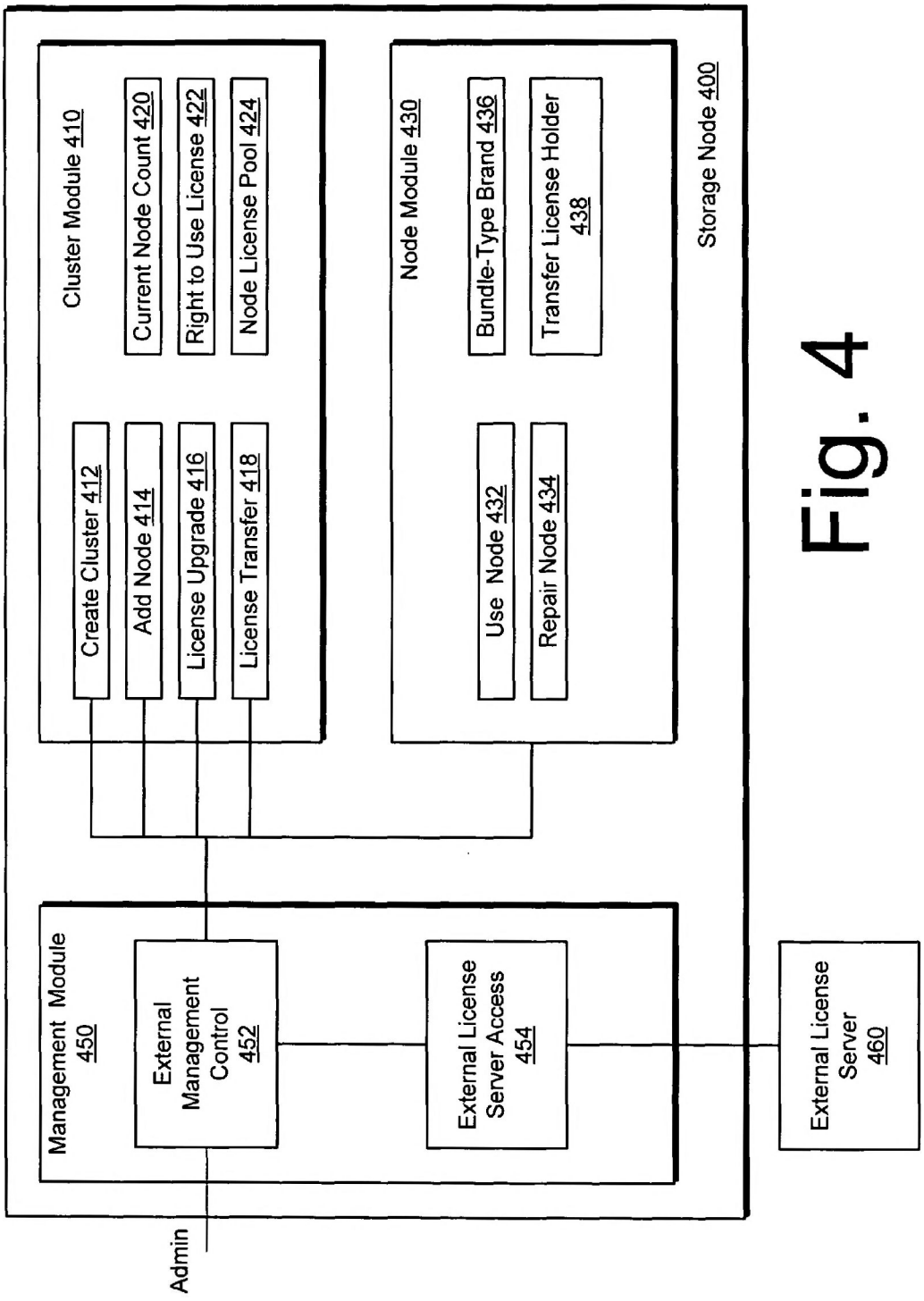

Storage Node 400
Cluster Module 410
Create Cluster 412
Add Node 414
License Upgrade 416
License Transfer 418
Current Node Count 420
Right to Use License 422
Node License Pool 424
Node Module 430
Use Node 432
Repair Node 434
Bundle-Type Brand 436
Transfer License Holder 438
Management Module 450
External Management Control 452
External License Server Access 454
External License Server 460
Admin

Fig. 4

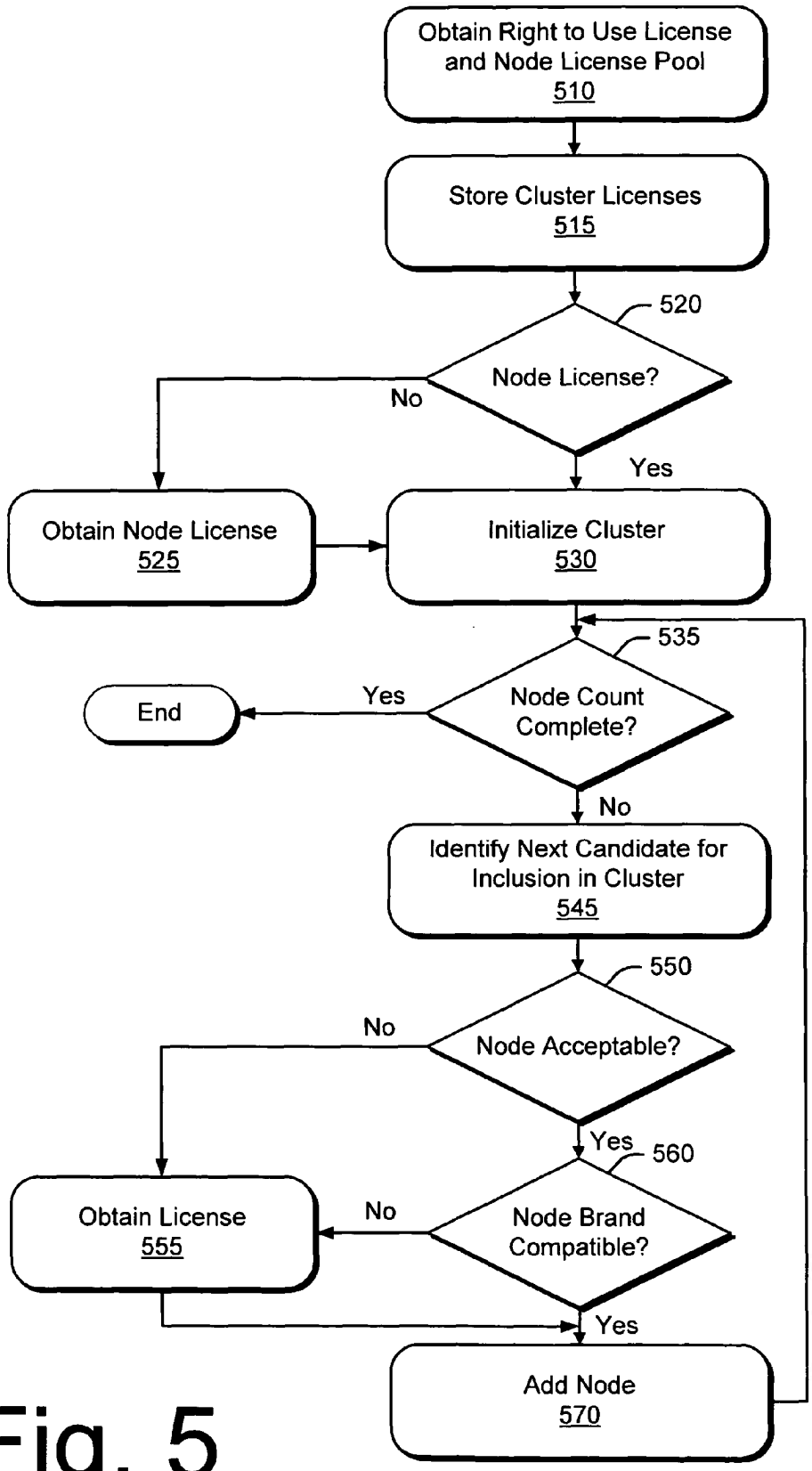

Obtain Right to Use License and Node License Pool
510
Store Cluster Licenses
515
520
Node License?
No
Yes
Obtain Node License
525
Initialize Cluster
530
535
Node Count Complete?
Yes
End
No
Identify Next Candidate for Inclusion in Cluster
545
550
Node Acceptable?
No
Yes
560
Node Brand Compatible?
No
Obtain License
555
Yes
Add Node
570

Fig. 5

Case 2:25-cv-00516 Document 1-1 Filed 05/09/25 Page 9 of 15 PageID #: 83

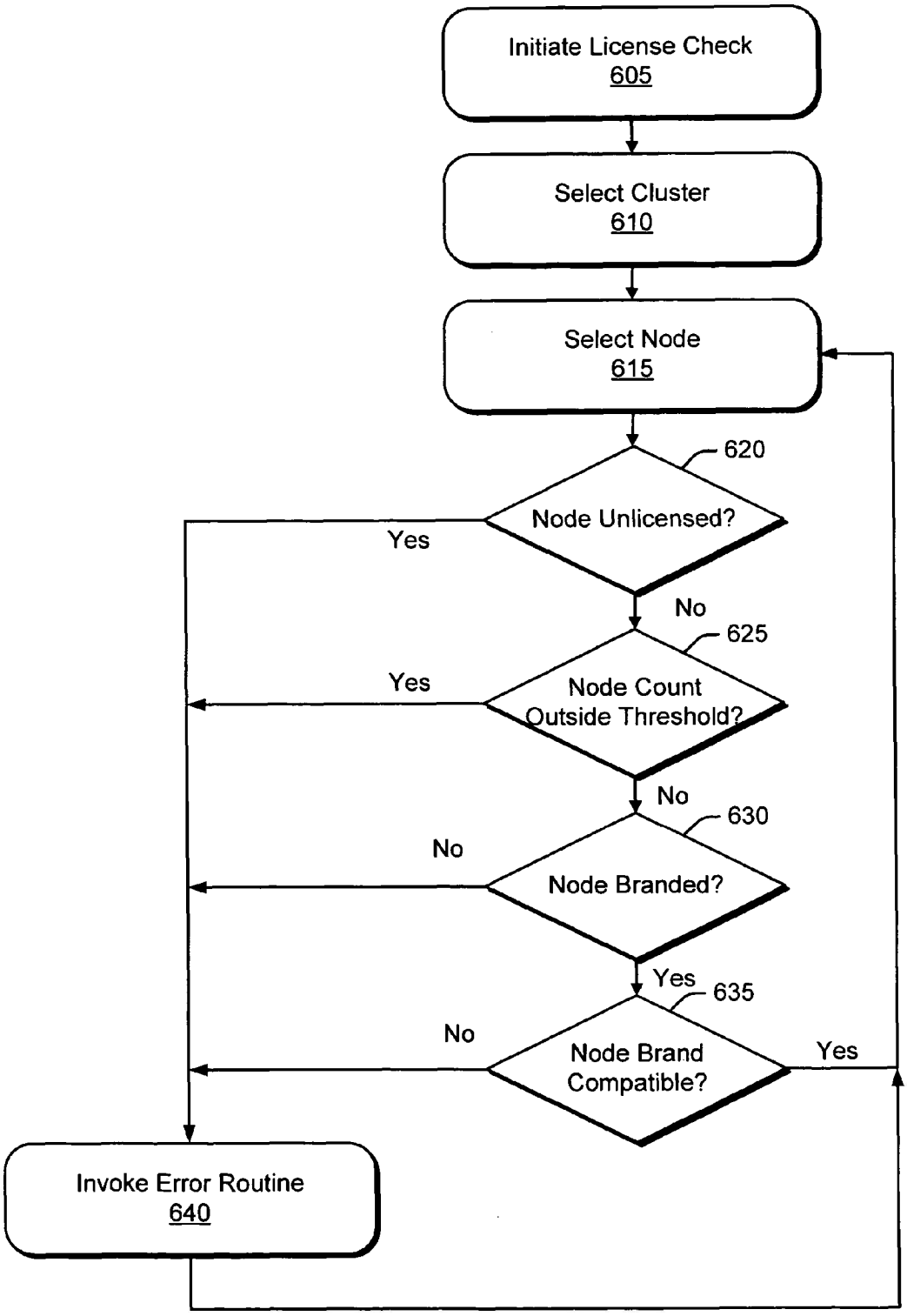


Fig. 6

COMPATIBILITY ENFORCEMENT IN CLUSTERED COMPUTING SYSTEMS

TECHNICAL FIELD

This application relates to electronic computing, and more particularly to compatibility enforcement in clustered computing systems.

BACKGROUND

The term "clustered computing system" refers to a computing system formed from a group of loosely coupled computing devices that cooperate to provide the appearance that the devices function as a single system. Clustering technology permits users to construct a variety of computing system configurations using elemental building blocks, and to change configurations or components of a clustered system as part of maintenance or update routines. By way of example, a high-capacity storage system may be constructed from a cluster of individual storage components. Similarly, a grid computer may be constructed from a cluster of individual computing components.

Computing systems and products created using clustering technology present particular technical issues. Technical issues include ensuring compatibility between hardware, software and firmware operating on clustered devices and ensuring compliance with licensing terms associated with individual computing devices and with cluster configurations.

SUMMARY

In one embodiment a method of creating a clustered computing system, comprises storing license information for a computing cluster in a memory module associated with a computing node, wherein the licensing information includes a bundle-type parameter that identifies a characteristic of the cluster and one or more node license parameters that identifies a characteristic of a computing node, initializing the computing cluster in a first computing node, adding one or more available computing nodes to the computing cluster; and activating the computing cluster when the computing cluster includes a number of nodes that complies with the rules associated with the bundle-type parameter.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a schematic illustration of a networked computing system that utilizes a storage network.

FIG. **2** is a schematic illustration of physical elements of a storage network according to an embodiment.

FIG. **3** is a schematic illustration of a storage node according to an embodiment.

FIG. **4** is a schematic illustration of a logic architecture for compatibility enforcement in clustered computing systems according to an embodiment.

FIG. **5** is a flowchart illustrating operations in a method for creating a clustered computing system according to an embodiment.

FIG. **6** is a flowchart illustrating operations in a method for performing a license check in a clustered computing system according to an embodiment.

DETAILED DESCRIPTION

Described herein are exemplary systems and methods for compatibility enforcement in clustered computing systems.

Certain methods described herein may be embodied as logic instructions and associated data on a computer-readable medium. When executed on one or more processors, the logic instructions cause the processor(s) to be programmed as a special-purpose machine that implements the described methods. The processor(s), when configured by the logic instructions to execute the methods recited herein, may constitute structure for performing the described methods.

In embodiments, subject matter described herein may be implemented in a storage architecture that provides data storage at a system level, such as, e.g., a storage area network (SAN). In implementations described herein, computing systems that utilize storage are referred to as hosts. For example, a host may be a supercomputer processing large databases, a transaction processing server maintaining transaction records, and the like. Alternatively, the host may be a file server on a local area network (LAN) or wide area network (WAN) that provides storage services for an enterprise.

In a direct-attached storage solution, a host may include one or more disk controllers or RAID controllers configured to manage multiple directly attached disk drives. By contrast, in a SAN a host connects to the SAN in accordance via a high-speed connection technology such as, e.g., a fibre channel (FC) fabric in the particular examples.

A virtualized SAN architecture may comprise one or more storage cells, where each storage cell comprises a group of storage devices called a disk group. Each storage cell comprises parallel storage controllers coupled to the disk group. The storage controllers may be coupled to the storage devices using a fibre channel arbitrated loop connection, or through a network such as a fibre channel fabric or the like. The storage controllers may also be coupled to each other through point-to-point connections to enable them to manage the presentation of storage capacity to computers using the storage capacity.

Network architectures described herein represent a distributed computing environment such as, e.g., an enterprise computing system using a private SAN. However, the network architectures may be readily scaled upwardly or downwardly to meet the needs of a particular application.

FIG. **1** is a schematic illustration of one embodiment of a networked computing system **100** that utilizes a storage network. In one embodiment, storage pool **110** may be implemented as a virtualized storage pool as described in published U.S. Patent Application Publication No. 2003/0079102 to Lubbers, et al., or as described in U.S. Patent Application No. 2005/0172073 to Voigt, et al., the disclosures of which is incorporated herein by reference in its entirety.

A plurality of logical disks (also called logical units, LUs or sometimes LUNs) **112***a*, **112***b* may be allocated within storage pool **110**. Each LUN **112***a*, **112***b* comprises a contiguous range of logical addresses that can be addressed by host devices **120**, **122**, **124** and **128** by mapping requests from the connection protocol used by the host device to the uniquely identified LUN **112***a*, **112***b*. A host such as server **128** may provide services to other computing or data processing systems or devices. For example, client computer **126** may access storage pool **110** via a host such as server **128**. Server **128** may provide file services to client **126**, and may provide other services such as transaction processing services, email services, etc. Hence, client device **126** may or may not directly use the storage consumed by host **128**.

Devices such as wireless device **120**, and computers **122**, **124**, which also may serve as hosts, may logically couple directly to LUNs **112***a*, **112***b*. Hosts **120**-**128** may couple to multiple LUNs **112***a*, **112***b*, and LUNs **112***a*, **112***b* may be shared among multiple hosts. Each of the devices shown in

FIG. **1** may include memory, mass storage, and a degree of data processing capability sufficient to manage a network connection.

A LUN such as LUN **112***a*, **112***b* comprises one or more redundant stores (RStore) which are a fundamental unit of reliable storage. An RStore comprises an ordered set of physical storage segments (PSEGs) with associated redundancy properties and is contained entirely within a single redundant store set (RSS). By analogy to conventional storage systems, PSEGs are analogous to disk drives and each RSS is analogous to a RAID storage set comprising a plurality of drives.

The PSEGs that implements a particular LUN may be spread across any number of physical storage disks. Moreover, the physical storage capacity that a particular LUN **102** represents may be configured to implement a variety of storage types offering varying capacity, reliability and availability features. For example, some LUNs may represent striped, mirrored and/or parity-protected storage. Other LUNs may represent storage capacity that is configured without striping, redundancy or parity protection.

A SAN manager appliance **109** is coupled to a management logical disk set (MLD) **111** which is a metadata container describing the logical structures used to create LUNs **112***a*, **112***b*, Logical Device Allocation Domains (LDADs) **103***a*, **103***b*, and other logical structures used by the system. A portion of the physical storage capacity available in storage pool **110** is reserved as quorum space **113** and cannot be allocated to LDADs **103***a*, **103***b*, and hence cannot be used to implement LUNs **112***a*, **112***b*. In a particular example, each physical disk that participates in storage pool **110** has a reserved amount of capacity (e.g., the first "n" physical sectors) that may be designated as quorum space **113**. MLD **111** is mirrored in this quorum space of multiple physical drives and so can be accessed even if a drive fails. In a particular example, at least one physical drive is associated with each LDAD **103***a*, **103***b* includes a copy of MLD **111** (designated a "quorum drive SAN management appliance **109** may wish to associate information such as name strings for LDADs **103***a*, **103***b* and LUNs **112***a*, **112***b*, and timestamps for object birthdates. To facilitate this behavior, the management agent uses MLD **111** to store this information as metadata.

Quorum space **113** may be used to store information including physical store ID (a unique ID for each physical drive), version control information, type (quorum/non-quorum), RSS ID (identifies to which RSS this disk belongs), RSS Offset (identifies this disk's relative position in the RSS), Storage Cell ID (identifies to which storage cell this disk belongs), PSEG size, as well as state information indicating whether the disk is a quorum disk, for example. This metadata PSEG also contains a PSEG free list for the entire physical store, probably in the form of an allocation bitmap. Additionally, quorum space **113** contains the PSEG allocation records (PSARs) for every PSEG on the physical disk. The PSAR comprises a PSAR signature, Metadata version, PSAR usage, and an indication a RSD to which this PSEG belongs.

CSLD **114** is another type of metadata container comprising logical drives that are allocated out of address space within each LDAD **103***a*, **103***b*, but that, unlike LUNs **112***a*, **112***b*, may span multiple LDADs **103***a*, **103***b*. Preferably, each LDAD **103***a*, **103***b* includes space allocated to CSLD **114**. CSLD **114** holds metadata describing the logical structure of a given LDAD **103**, including a primary logical disk metadata container (PLDMC) that contains an array of descriptors (called RSDMs) that describe every RStore used by each LUN **112***a*, **112***b* implemented within the LDAD **103***a*, **103***b*. The CSLD **114** implements metadata that is regularly used for tasks such as disk creation, leveling, RSS merging, RSS splitting, and regeneration. This metadata includes state information for each physical disk that indicates whether the physical disk is "Normal" (i.e., operating as expected), "Missing" (i.e., unavailable), "Merging" (i.e., a missing drive that has reappeared and must be normalized before use), "Replace" (i.e., the drive is marked for removal and data must be copied to a distributed spare), and "Regen" (i.e., the drive is unavailable and requires regeneration of its data to a distributed spare).

A logical disk directory (LDDIR) data structure in CSLD **114** is a directory of all LUNs **112***a*, **112***b* in any LDAD **103***a*, **103***b*. An entry in the Logical Disk Directory Structure (LDDS) comprises a universally unique ID (UUID) an RSD indicating the location of a Primary Logical Disk Metadata Container (PLDMC) for that LUN **102**. The RSD is a pointer to the base RSDM or entry point for the corresponding LUN **112***a*, **112***b*. In this manner, metadata specific to a particular LUN **112***a*, **112***b* can be accessed by indexing into the LDDIR to find the base RSDM of the particular LUN **112***a*, **112***b*. Metadata within the PLDMC (e.g., mapping structures described herein) can be loaded into memory to realize the particular LUN **112***a*, **112***b*.

Hence, the storage pool depicted in FIG. **1** implements multiple forms of metadata that can be used for recovery. The CSLD **111** implements metadata that is regularly used for tasks such as disk creation, leveling, RSS merging, RSS splitting, and regeneration. The PSAR metadata held in a known location on each disk contains metadata in a more rudimentary form that is not mapped into memory, but can be accessed when needed from its known location to regenerate all metadata in the system.

Each of the devices shown in FIG. **1** may include memory, mass storage, and a degree of data processing capability sufficient to manage a network connection. The computer program devices in accordance with the present invention are implemented in the memory of the various devices shown in FIG. **1** and enabled by the data processing capability of the devices shown in FIG. **1**.

FIG. **2** is a schematic illustration of physical elements of a storage network **200** that may be used to implement a storage network such the network depicted in FIG. **1**. Storage network **200** comprises a plurality of clusters **230**, **240**, **250**, which may be implemented as one or more communicatively connected storage nodes. Thus, cluster **230** includes storage nodes **232**, **234**, **236**. Cluster **240** includes nodes **242**, **244**, **246**. Cluster **250** includes nodes **252**, **254**, **256**. Each node may include one or more storage devices such as, e.g., an array of storage devices. Exemplary storage devices include the STORAGEWORKS line of storage devices commercially available form Hewlett-Packard Corporation of Palo Alto, Calif., USA.

Network **200** may further include one or more host computers **210** coupled to clusters **230**, **240**, **250** via a communication network **220**. Communication network **220** may be implemented as a private, dedicated network such as, e.g., a Fibre Channel (FC) switching fabric. Alternatively, portions of communication network **220** may be implemented using public communication networks pursuant to a suitable communication protocol such as, e.g., the Internet Small Computer Serial Interface (iSCSI) protocol.

One or more client computers **212** may access resources provided by storage clusters **230**, **240**, **250** through host **210**. In operation, one or more client computers **212** generate a request to host computer **210** for a resource provided by one or more of storage clusters **230**, **240**, **250**. Host computer **210** generates a resource request to one or more of clusters **230**, **240**, **250** for the requested resource(s). One or more of clus-

ters **230**, **240**, **250** process the resource request, and return the requested resource(s) to host **210**, which in turn forwards the requested resource to client computer(s) **212**.

Storage network **200** may further comprise a compatibility server **214** that maintains compatibility information relating to components of storage network **200**. Compatibility information may include information relating to, e.g., firmware versions and/or revisions, and licensing information.

FIG. **3** is a schematic illustration of an exemplary implementation of a storage node **300** that may be used in a cluster such as cluster **230**, **240**, **250**. Referring to FIG. **3**, in one embodiment each storage node **300** includes one or more Network Storage Controllers (NSCs) **310**, also referred to as disk array controllers or storage controllers, to manage the operations and the transfer of data to and from one or more disk drives **340**, **342**. NSCs **310** may be implemented as plug-in cards having a microprocessor **316**, and memory **318**. Each NSC **310** includes one or more host adapter ports **312**, **314** that provide an interface to a host such as host **210**, i.e., through a communication network such as a switching fabric. In a Fibre Channel implementation, host adapter ports **312**, **314**, may be implemented as FC N_Ports. Each host adapter port **312**, **314** manages the login and interface with a switching fabric, and is assigned a fabric-unique port ID in the login process.

In an exemplary implementation, NSC **310** further includes one or more Fiber Channel Arbitrated Loop (FCAL) ports **320**, **322**, **234**, **326** that implement an FCAL communication connection with a plurality of storage devices, e.g., arrays of disk drives **340**, **342**. While the illustrated embodiment implement FCAL connections with the arrays of disk drives **340**, **342**, it will be understood that the communication connection with arrays of disk drives **340**, **342** may be implemented using other communication protocols. For example, rather than an FCAL configuration, a FC switching fabric or a small computer serial interface (SCSI) connection may be used.

In embodiments, a clustered computing system such as the clustered storage devices depicted in FIGS. **1-3** may be adapted to implement compatibility enforcement between components in the system. As used herein, compatibility enforcement may include compatibility between hardware, software, firmware, and/or licensing requirements. In one embodiment, a clustered computing system implements a licensing scheme that ensures licensing compliance at both the component level and at the system level. The licensing scheme may implement a multi-tiered (or bundled) license program. The first part of the license scheme defines a permissible size range for a cluster. In one embodiment, clusters may be categorized into multiple size ranges, referred to herein as bundle-types. In alternate embodiments, clusters may be categorized into bundle types by one or more additional parameters such as, e.g., an array, tape, computing, translating functional bundle. For example, a bundle may refer to a sales bundle. A bundle may be sold as a set or group of node or grid types to meet a specific market or function. For example, a product category may include one or more bundle sizes that served a single purpose such as, e.g., array/LUN purpose. Bundles may have a specific homogeneous purpose along with standard grid parameters such as, e.g., right to use, maximum and minimum node counts, and a specific array type functional brand. Other bundles may have different characteristics within those listed above, for example, the purpose or function. A bundle may have multipurpose or multiple different single purposed nodes in one bundle. The second part of the license scheme permits the cluster to permanently associate a license and bundle-type with a defined number of nodes as they join the cluster.

FIG. **4** is a schematic illustration of a logic architecture for compatibility enforcement in clustered computing systems according to an embodiment. In one embodiment, the clustered computing system may correspond to one or more of the storage nodes in the clusters **230**, **240**, **250** of FIG. **2**. The logic modules in storage node **400** may be embodied as software or firmware stored in a computer readable medium such as, e.g., the memory **318** of a storage array controller **300** and executable by a processor such as microprocessor **316** of array controller **300**.

Referring to FIG. **4**, storage node **400** includes a cluster module **410** that includes logic to manage cluster-level compatibility functions. Cluster module **410** includes a create cluster module **412**, an add node module **414**, a license upgrade module **416**, and a license transfer module **418** which permits removal of a node that is not part of a base bundle. Cluster module further includes a current node count data structure **420** that holds the current number of valid nodes in the cluster, a right to use license data structure **422**. In one embodiment, the right to use license may include a token that is conferred by a license management system. The token may include license management specific information for authenticating the license and an indication of a bundle type and the maximum or minimum number of nodes that can be included in the cluster. Cluster module **420** further includes a node license pool **424**. In one embodiment, node license pool **424** comprises a pool of tokens conferred on the cluster by a license management system. Each token in the pool includes license management specific information for authenticating the license. The tokens are consumed as they are assigned to nodes to enable nodes to join the cluster. Tokens may be associated with a unique identifier for the node that has been assigned each node license by the cluster.

Storage node **400** further includes a node module **430** that includes logic to manage node-level compatibility functions. Node module **430** includes a use node **432** and a repair node **434**. Node module **432** further includes a bundle-type data structure **436** that holds a token that indicates the node's membership in a valid cluster and the cluster's bundle-type. A brand may beis obtained when a node successfully joins a correctly licensed cluster.

Node module **432** may further include a transfer license holder data structure **438**. In one embodiment, transfer license holder data structure **438** may provide a temporary location for a Right-to-Use License and a Node License pool associated with a cluster that is decommissioned. The transfer license holder data structure provides a way to reconfigure a clustered computing system without having to re-enter license keys. Algorithms associated with the Transfer License Holder insure that transfer cannot result in more than one usable copy of the license.

FIG. **5** is a flowchart illustrating operations in a method for creating an initial clustered computing system according to an embodiment. Referring to FIG. **4** and FIG. **5**, at operation **510** a Right-to-Use License **422** and a Node License pool **424** are established, for example via the initial bundle parameters as determined based on which the bundle is to satisfy. Right-to-Use and Node license(s) may be acquired using a key provided through initial sale of the bundle that is entered through external management control software. Referring briefly to FIG. **4**, in one embodiment, a management module **450** may include an external management control module **452** and an external license server access module **454** that provide an interface with an external license server **460**. A license key

may be passed to a license server via the external license server access software. The license server returns a detailed right to use license **422** and Node License pool **424** information associated with the bundle key. A node that was not part of the original bundle SKU may be included in an initial cluster If the node has a license that posses a characteristic known as an "Add-on". This licenses characteristic may also be obtained via the license server.

At operation **515** the license clusters, which may correspond to the pools described above, are stored. In one embodiment, the Right-to-Use License and the Node License pool may be stored in the transfer license holder data structure in the node which originated the license request. The licensing information may be stored in the memory **318** of one or more array controllers **310** in a cluster **230**, **240**, **250**. The memory module **318** may be a non-volatile memory such that the licensing information remains intact in the event that the array controller **310** loses power or fails.

If, at operation **520**, the node that initiated the license request does not have a node license of its own and there are licenses left in the node license pool, then one of the node licenses may be conferred to the node receiving the management command. Operation **525** may be done in two phases. In the first phase the node license pool may be updated to associate a unique identifier (e.g., a serial number) with the node that is assigned the license. In the second phase, the node may be "branded" as a licensed node having a specific bundle type, e.g., by writing a token to a protected non-volatile memory location on the node. The node license may then be assigned specifically to the node (and may not be assigned to any other node). Control may then pass to operation **530**, described below.

By contrast, if at operation **520** the node has a node license that matches one or more required parameters, then control passes to operation **530** and the cluster may be initialized. In one embodiment, the cluster may be initialized with as few as one member, i.e., the node that received the management command to create the cluster. The Right-to-Use License and the node license pool may be transferred to the newly created single node cluster, a node count parameter may be initialized (i.e., set to 1).

If, at operation **535**, the node count for the cluster has reached a limit specified in the Right-to-Use license (or if there are no more candidate nodes), the cluster formation process is complete and the process ends. By contrast, if at operation **535** the node count for the cluster has not been reached. (i.e., if the cluster has not yet activated a number of nodes authorized by the license terms), then control passes to operation **545** and a next candidate node for inclusion in the cluster may be identified. This can be done using any number of techniques common in the art.

At operation **550** the candidate node's acceptability is evaluated. In one embodiment evaluating the node's acceptability at operation **550** encompasses operations **620**-**635**, described with reference to FIG. **6**. If the candidate node is unacceptable (i.e., if the node is new or otherwise fails an acceptability test), then control passes to operation **555** and a license is obtained for the candidate node. In one embodiment, a license may be obtained as described in connection with operation **525**. In the event that a license cannot be obtained an error routine may be invoked.

By contrast, if at operation **550** the node is acceptable, then control passes to operation **560** and it is determined whether the node's brand is compatible with the cluster as defined by the Right-To-Use characteristics. In one embodiment, if the node has a brand which indicates that the node is no longer part of a different cluster (i.e., if the node has been released from a different cluster) and if its brand is valid for inclusion in this cluster's bundle-type (for example, lower class bundle types cannot join a cluster with a higher class bundle-type, but the reverse is true, e.g., higher class nodes can join lower class clusters), then no additional node license needs to be conferred on the node from the cluster's node license pool. This embodiment allows nodes to be moved between clusters provided both node and cluster bundle-types are compatible.

In another embodiment the candidate node may require an additional license from the cluster's node license pool. In this embodiment, movement of nodes between clusters requires re-licensing the node such as upgrade from a lower-class to a higher-class node. Hence control may pass to operation **555** and the candidate node may be licensed.

Control then passes to operation **570** and the candidate node may be added to the cluster. Control then returns to operation **535**. Thus, operations **535**-**570** define a loop which may be repeated to construct a clustered computing system. In one embodiment a final test may be performed to ensure that the cluster includes the minimum number of nodes before initializing the cluster services and proceed with normal operations. This closed-loop precludes adding nodes that have a questionable lineage such as, e.g., gray market nodes.

FIG. **6** is a flowchart illustrating operations in a method for performing a license check in a clustered computing system according to an embodiment. In one embodiment, a license check may be executed on a periodic basis. In alternate embodiment, a license check may be event driven. For example, a license check may be implemented when a cluster bootstraps to insure that all nodes are properly licensed for the cluster.

Referring to FIG. **6**, a license check is initiated at operation **605**, at operation **610** a first cluster is selected, and at operation **615** a first node in the cluster is selected. If, at operation **620**, the selected node is not properly licensed, e.g., if a node's license is expired or if the node brand is incompatible with type of cluster in which the node is placed, then control passes to operation **640** and an error routine may be invoked. In one embodiment, the error routine may include generating a warning signal that may be displayed on a user interface such as, e.g., a display or the like. In another embodiment, the error routine may include disabling the node, or disabling the entire cluster. By contrast, if at operation **620** the node is properly licensed, then control passes to operation **625**.

If, at operation **625**, the node count in the cluster is outside a threshold, then control passes to operation **640** and the error routine is invoked as described above. In one embodiment operation **625** also happens in the initial cluster generation and subsequently with every newly attempted node addition. In one embodiment, the threshold may include at least one of a lower limit on a number of nodes in a cluster and an upper limit on a number of nodes in a cluster. By contrast, if at operation **625**, the node count in the cluster is within the threshold values, then control passes to operation **630**.

If, at operation **630**, a node is not branded, then control passes to operation **640** and the error routine is invoked as described above. In one embodiment, an unbranded node may indicate that the node is new and needs to be licensed, either through a cluster creation process or through a process of adding a node to an existing cluster. By contrast, if at operation **630**, the node is branded, then control passes to operation **635**.

If, at operation **635**, a node brand is not compatible with the brand of the cluster, then control passes to operation **640** and the error routine is invoked as described above. For example, a node may be branded to work only within clusters having fewer than a specific number of nodes. By contrast, if at

operation **635**, the node brand is compatible with the cluster brand, then control passes to operation **615** and another node is selected. Thus, operations **615-635** define a loop by which nodes in a cluster may be evaluated for compliance with licensing and/or compatibility enforcement terms. The operation terminates if, at operation **615**, there are no more nodes to evaluate.

A cluster may be expanded by adding one or more nodes to the cluster. In one embodiment, nodes may be added to a cluster using the operations **545-570** described in FIG. **5**. The cluster must have an unassigned node license in order for a node with no bundle-type (i.e. a new, un-initialized node) to be added. The unassigned license may be an Add-on license purchased with an Add-on node, or an unassigned base bundle license that came with the original SKU.

A node may be unable to retain its bundle-type brand due to, e.g., a hardware failure or the like. In one embodiment, the repair node module **434** permits a user such as an administrator to reinstate a node license lost due to failure. Repair node module **434** provides the ability to cross-reference node licenses in the cluster against the cluster's original license(s). This generally involves access to the license server.

A cluster's Right-to-Use license **422** may be upgraded, e.g., by launching an upgrade inquiry with the external license server **460**. When this occurs the Right-to-Use license can be updated to reflect new characteristics such as maximum and minimum node count thresholds, and additional node licenses can be added to the node license pool. Similarly, bundle-type brands can be upgraded on nodes given that the necessary fees have been paid and the licenses can be authenticated. After either of these upgrades operations **545-570** of the cluster creation process may be repeated to account for nodes that can be added to the cluster.

When a cluster is operational its license information may be stored redundantly on multiple nodes within the cluster. When a cluster is de-commissioned its license information may be stored in a node's transfer license holder **438** as a convenient way of transferring the licensing information to a new cluster. In one embodiment, the node which stores the license information may be the initial node in the new cluster. Hence, the cluster creation can effectively begin at operation **530**. If the node fails so that the transfer license is lost, then the node may be reinstated using node repair techniques.

Reference in the specification to "one embodiment" or "an embodiment" means that a particular feature, structure, or characteristic described in connection with the embodiment is included in at least an implementation. The appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same embodiment.

What is claimed is:

1. A method of creating a clustered computing system, comprising:

storing license information for a computing cluster in a memory module associated with a computing node, wherein the licensing information includes a bundle-type parameter that identifies a size of the cluster and at least one additional bundle-type characteristic of the cluster, and one or more node license parameters that identifies a characteristic of a computing node;

initializing the computing cluster in a first computing node;

adding one or more available computing nodes to the computing cluster; and

activating the computing cluster when the computing cluster includes a number of nodes that complies with the bundle-type parameter.

2. The method of claim **1**, wherein storing license information for a computing cluster in a memory module associated with a computing node comprises retrieving licensing information from an external licensing server.

3. The method of claim **1**, wherein the bundle-type parameter identifies at least one of a minimum or maximum node count threshold for the computing cluster.

4. The method of claim **1**, wherein: the node license parameters identify one or more bundle-types compatible with the computing node.

5. The method of claim **1**, wherein adding one or more available computing nodes to the computing cluster comprises assigning a computing node a new node license when the computing node license is incompatible with a brand-type associated with the computing cluster.

6. The method of claim **1**, further comprising initiating a license check while the computing system is operating, wherein the license check evaluates at least one of: a license status of the computing node; a node count in the computing cluster; and a brand-type associated with the computing node.

7. The method of claim **1**, wherein the license check comprises implementing an error routine that inhibits operation of the computing node when an incompatibility between the computing node and the computing cluster is detected.

8. The method of claim **1**, further comprising adding a computing node to the activated computing cluster.

9. A computing system comprising at least two computing nodes, wherein at least one of the computing nodes comprises:

a cluster module stored on a non-transitory computer readable medium, the cluster module configured to manage compatibility operations associated with a computing cluster, the cluster module initializing a computing cluster in the first computing node, the cluster module adding one or more available computing nodes to the computing cluster;

a node module stored on a non-transitory computer readable medium, the node module configured to manage compatibility operations associated with the at least two computer nodes, the node module storing license information for the computing cluster, wherein the licensing information includes a bundle-type parameter that identifies a size of the computing cluster and at least one additional bundle-type characteristic of the computing cluster, and one or more node license parameters that identifies a characteristic of a first computing node; and

a management module stored on a non-transitory computer readable medium, the management module configured to manage compatibility operations with one or more external servers, the management module activating the computing cluster when the computing cluster includes a number of nodes that complies with the bundle-type parameter.

10. The computing system of claim **9**, wherein the cluster module comprises a create cluster module to aggregate the at least two computing nodes into a clustered computing system.

11. The computing system of claim **10**, wherein the cluster module maintains a right to use license comprising licensing information for a computing cluster and a node license pool comprising licensing information for nodes in the computing cluster.

12. The computing system of claim **10**, further comprising an add node module to add a computing node to the clustered computing system.

13. The computing system of claim 10, wherein the management module provides a license key to and receives licensing information from an external licensing server.

14. A computer program product comprising logic instructions stored on a non-transitory computer-readable medium which, when executed by a computer processor, configure the processor to:

store license information for a computing cluster in a memory module associated with a computing node, wherein the licensing information includes a bundle-type parameter that identifies a size of the cluster and at least one additional bundle-type characteristic of the cluster, and one or more node license parameters that identifies a characteristic of a computing node;

initialize the computing cluster in a first computing node;

add one or more available computing nodes to the computing cluster; and

activate the computing cluster when the computing cluster includes a number of nodes that complies with the bundle-type parameter.

15. The computer program product of claim 14, further comprising logic instructions which, when executed by the processor, configure the processor to retrieve licensing information from an external licensing server.

16. The computer program product of claim 14, further comprising logic instructions which, when executed by the processor, configure the processor to assign a computing node a new node license when the computing node license is incompatible with a brand-type associated with the computing cluster.

17. The computer program product of claim 14, further comprising logic instructions which, when executed by the processor, configure the processor to initiate a license check while the computing system is operating, wherein the license check evaluates at least one of:

a license status of the computing node;

a node count in the computing cluster; and

a brand-type associated with the computing node.

18. The computer program product of claim 14, further comprising logic instructions which, when executed by the processor, configure the processor to implement an error routine that inhibits operation of the computing node when an incompatibility between the computing node and the computing cluster is detected.

* * * * *

EXHIBIT B

US007904686B2

(12) **United States Patent**
Iyengar

(10) **Patent No.: US 7,904,686 B2**
(45) **Date of Patent: Mar. 8, 2011**

(54) **DATA SECURITY FOR USE WITH A FILE SYSTEM**

(75) Inventor: **Ranganath G. Iyengar**, Kamataka (IN)

(73) Assignee: **Hewlett-Packard Development Company, L.P.**, Houston, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 346 days.

(21) Appl. No.: **12/152,595**

(22) Filed: **May 13, 2008**

(65) **Prior Publication Data**

US 2009/0327635 A1 Dec. 31, 2009

(30) **Foreign Application Priority Data**

Jul. 31, 2007 (IN) 1671/CHE/2007

(51) **Int. Cl.**
G06F 12/00 (2006.01)

(52) **U.S. Cl.** **711/163**

(58) **Field of Classification Search** ..... None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,024,583	B2	4/2006	Nguyen et al.	
7,430,585	B2 *	9/2008	Sibert	709/216
2003/0120938	A1 *	6/2003	Mullor	713/190
2004/0153718	A1 *	8/2004	Shen et al.	714/5
2006/0080520	A1 *	4/2006	Dickenson	711/170
2007/0250719	A1 *	10/2007	Lai et al.	713/190
2008/0022396	A1 *	1/2008	Kado	726/19
2008/0034264	A1 *	2/2008	Kaabouch et al.	714/732

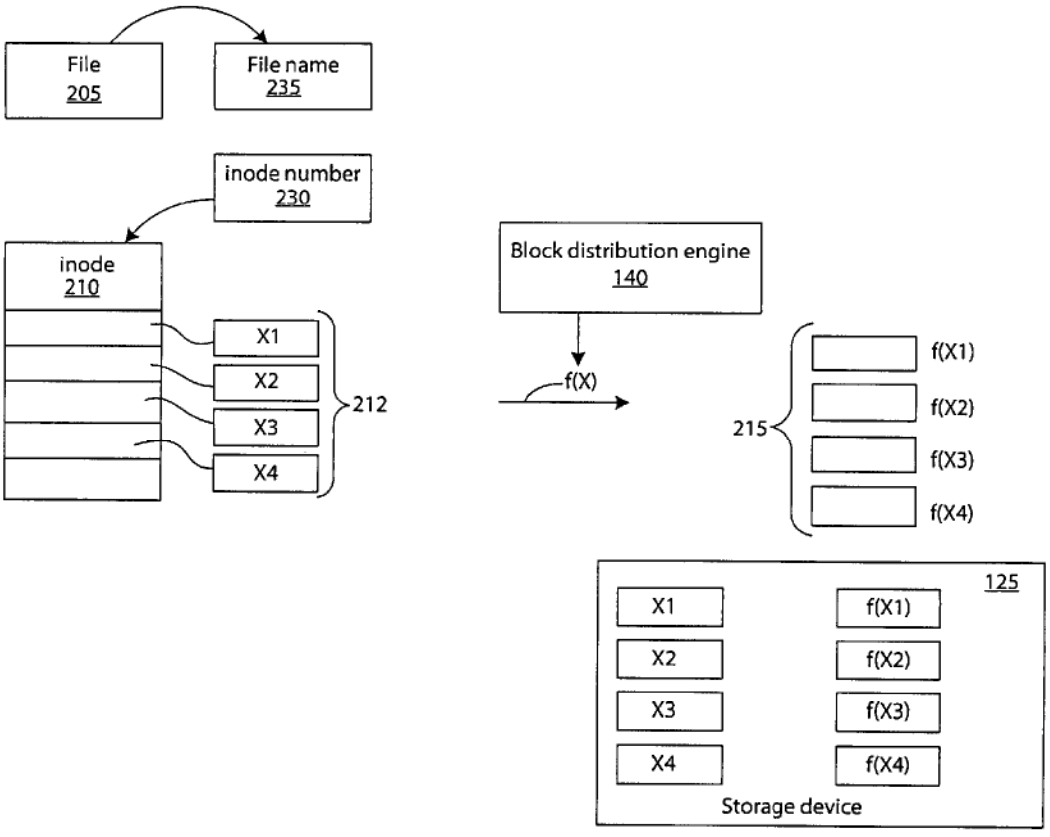
* cited by examiner

Primary Examiner — Hiep T Nguyen

(57) **ABSTRACT**

An embodiment of the invention provides an apparatus and method for providing data security for use with a file system. The apparatus and method performs acts including: applying a mapping function to data block numbers that are associated with a file; and obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.

18 Claims, 3 Drawing Sheets



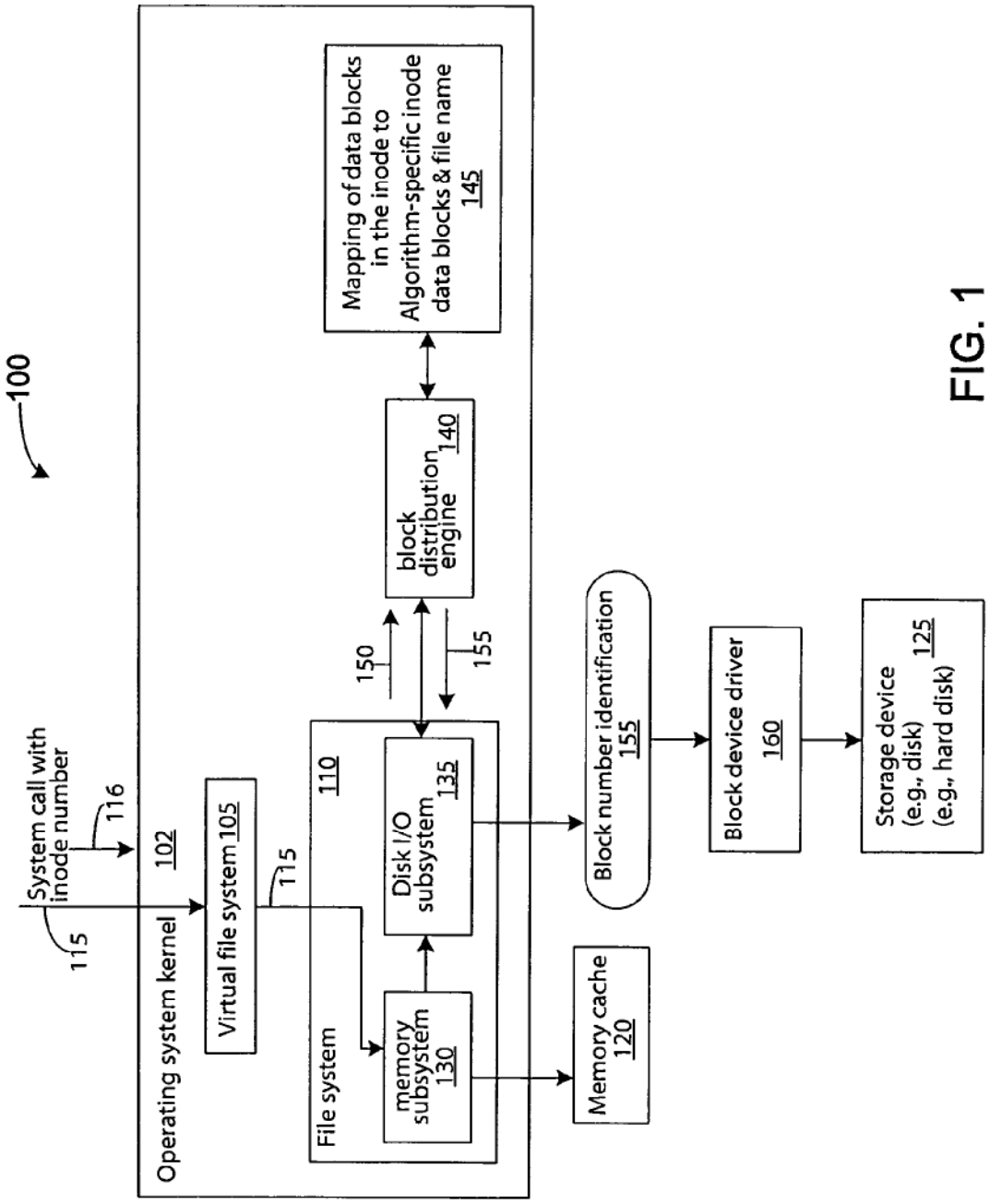

100
System call with inode number
116
115
102
Operating system kernel
Virtual file system 105
115
110
File system
memory subsystem 130
Disk I/O subsystem 135
150
155
block distribution engine 140
Mapping of data blocks in the inode to Algorithm-specific inode data blocks & file name 145
Memory cache 120
Block number identification 155
Block device driver 160
Storage device (e.g., disk) (e.g., hard disk) 125

FIG. 1

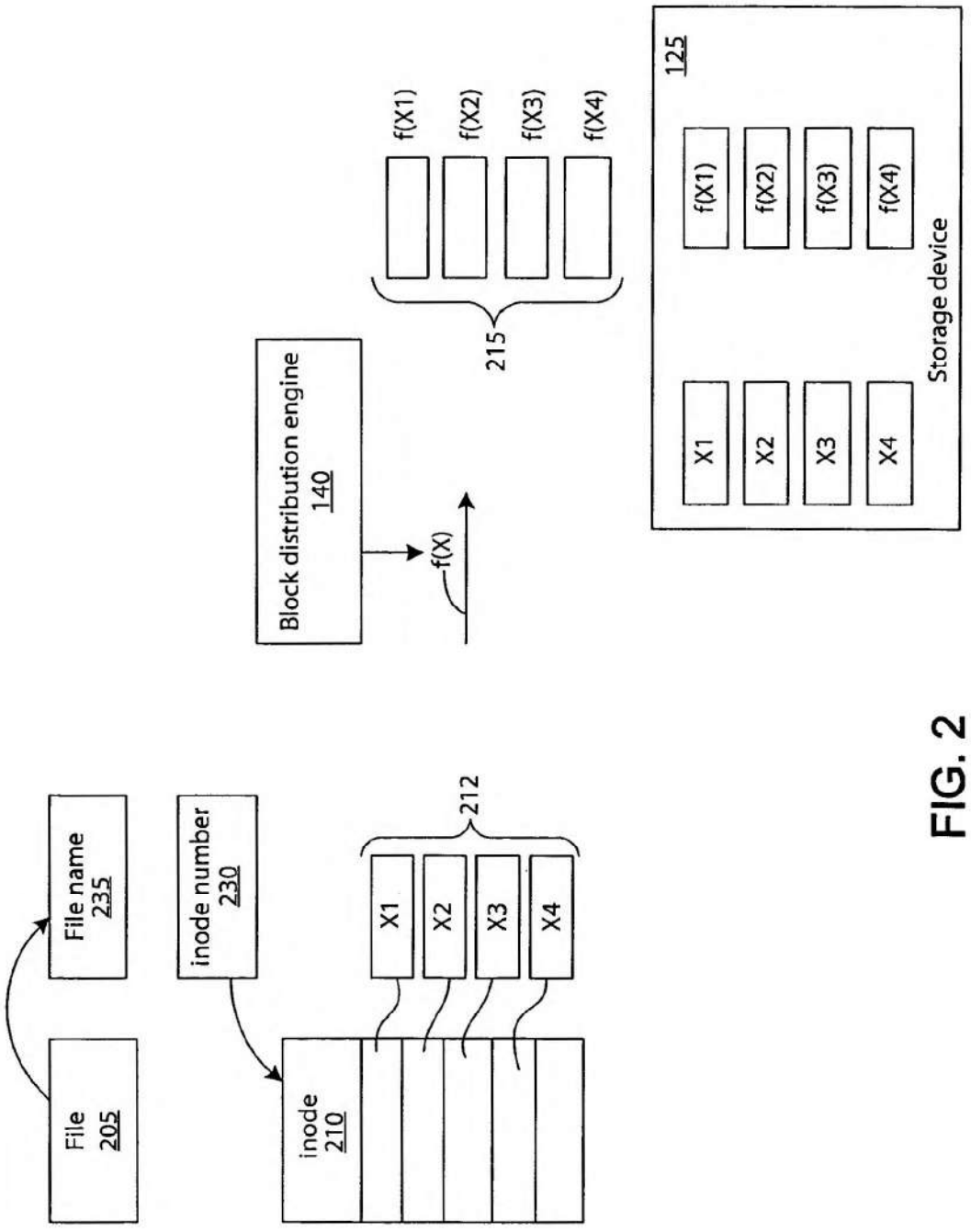

File
205
File name
235
inode number
230
inode
210
212
X1
X2
X3
X4
Block distribution engine
140
f(X)
f(X1)
f(X2)
f(X3)
f(X4)
215
125
Storage device
X1
X2
X3
X4
f(X1)
f(X2)
f(X3)
f(X4)

FIG. 2

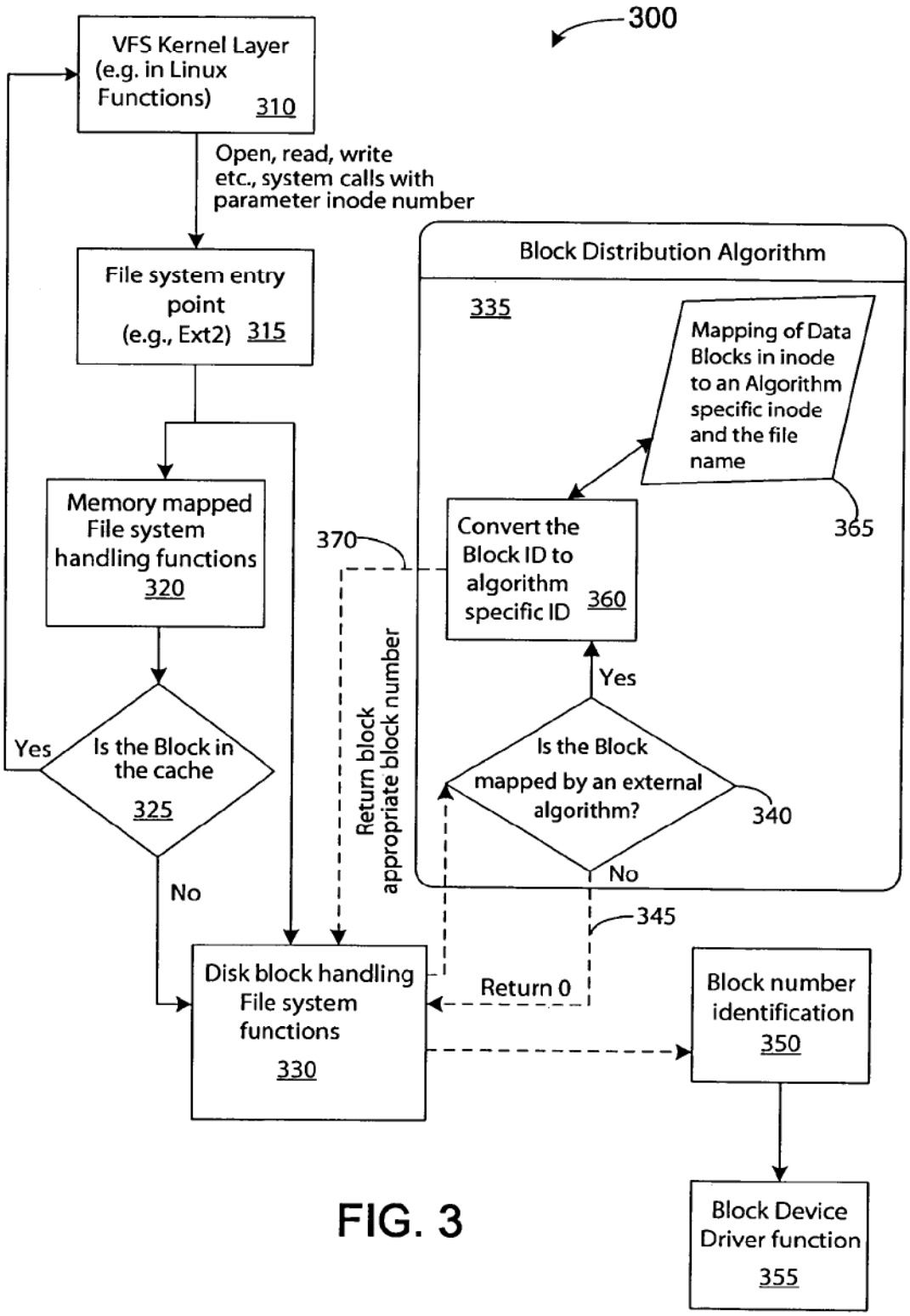

300
VFS Kernel Layer (e.g. in Linux Functions) 310
Open, read, write etc., system calls with parameter inode number
File system entry point (e.g., Ext2) 315
Block Distribution Algorithm
335
Mapping of Data Blocks in inode to an Algorithm specific inode and the file name
365
Memory mapped File system handling functions 320
370
Convert the Block ID to algorithm specific ID 360
Return block appropriate block number
Yes
Is the Block mapped by an external algorithm? 340
No
Yes
Is the Block in the cache 325
No
345
Disk block handling File system functions 330
Return 0
Block number identification 350
Block Device Driver function 355

FIG. 3

DATA SECURITY FOR USE WITH A FILE SYSTEM

TECHNICAL FIELD

Embodiments of the invention relate generally to data security for use with a file system.

BACKGROUND

A current method of providing security to data (e.g., software or other types of files) that is stored in a storage device (e.g., compact disk) so that unauthorized copying of the data is prevented is to provide encryption for the data. Therefore, access to the data is not permitted unless the user has the proper decryption code for the data. However, encryption-decryption software applications are typically time consuming to use and have often caused confusion for the users.

Online licensing checks are performed for some licensed software files to prevent un-authorized copying of software. However, this approach requires the software developer to embed the licensing code in each software file and requires the purchaser to access a public network (e.g., Internet) for the online licensing check.

Furthermore, even if the data in the storage device is subject to a licensing check, an individual can still reconstruct a file (e.g. software) on a different disk, without the need for proper authorization, by retrieving the data blocks that make up the file (or set of files). For example, on a UNIX file system, an individual can retrieve the data blocks of a file by accessing the index node (inode) of the file and completely bypass the application that is used for accessing the file.

On the other hand, if security is not provided to the stored data, then individuals will be able to make unauthorized copies of software or other files that are stored in storage devices (e.g., CDs).

Therefore, the current technology is limited in its capabilities and suffers from at least the above constraints and deficiencies.

BRIEF DESCRIPTION OF THE DRAWINGS

Non-limiting and non-exhaustive embodiments of the present invention are described with reference to the following figures, wherein like reference numerals refer to like parts throughout the various views unless otherwise specified.

FIG. **1** is a block diagram of an apparatus (system) in accordance with an embodiment of the invention.

FIG. **2** is a block diagram that shows additional details of a data block mapping in accordance with an embodiment of the invention.

FIG. **3** is a flow diagram of a method in accordance with an embodiment of the invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

In the description herein, numerous specific details are provided, such as examples of components and/or methods, to provide a thorough understanding of embodiments of the invention. One skilled in the relevant art will recognize, however, that an embodiment of the invention can be practiced without one or more of the specific details, or with other apparatus, systems, methods, components, materials, parts, and/or the like. In other instances, well-known structures, materials, or operations are not shown or described in detail to avoid obscuring aspects of embodiments of the invention.

FIG. **1** is a block diagram of an apparatus (system) **100** in accordance with an embodiment of the invention. The apparatus **100** can be implemented in, for example, a computer. An operating system kernel **102** includes a virtual file system **105** and a file system **110**. It is understood that the operating system **102** includes known modules for performing OS management operations. A virtual file system **105** is typically included in an operating system kernel layer and allows client applications (not shown in FIG. **1**) to transparently access the different types of file systems **110** (e.g., Unix file systems, Windows® file systems, MAC® OS file systems) that may be implemented in the kernel **102**. However, an embodiment of the invention can also be implemented in a system without the virtual file system **105**.

As known to those skilled in the art, an inode (index node) is a data structure that contains information about a file, directory, or other object in a file system. The file can be, for example, a software application, or other types of data such as a WORD® document. The inode data structure is used in, for example, a Unix file system or other types of file systems. The file system **110** will assign a unique inode for each file. Each inode typically contains the following information: the device (e.g., disk) where the inode resides, locking information, mode and type of file, the number of links to the file, the owner's user and group identifiers, the number of bytes in the file, access and modification times of the file, the time the inode itself was last modified, and the addresses of the file's data blocks on the disk.

Assume that the file system **110** has stored a file into a memory such as, for example, a memory cache **120** (e.g., memory in a computer) or a storage device **125** (e.g., CD or hard disk). Since the file system **110** has stored the file, the file system **110** identifies the file by an inode number. The inode number is shown symbolically as arrow **116** in FIG. **1**. That inode number, in turn, will index into an inode table that contains the inode for the file. The inode contains the addresses of the file's data blocks on, for example, the storage device (e.g., disk) **125**. When a user opens, reads, and/or writes to the file, the system call **115** is processed by the virtual file system **105** and forwarded to the file system **110**. The system call **115** also includes the inode number of the file to be opened, read, and/or written.

Typically, the memory subsystem **130** will first check if the data blocks of the file is currently in the memory cache **120**. This checking step is an optimization step that increases the speed of opening of, reading of, and/or writing to a file by fetching the data blocks of the files in the memory cache **120**, instead of always performing a data block access to the storage device **125**. However, an embodiment of the invention may also be implemented in a system that omits the use of a memory subsystem **130** that checks the data blocks in the memory cache **120**.

If the data blocks of the file that is identified the inode number (arrow **116**) is not in the memory cache **120**, then the disk I/O (input/output) subsystem **135** of the file system **110** will handle the system call **115** with the inode number **116**. In previous systems, the disk I/O (input/output) subsystem **135** computes the block numbers of the file in order to determine the location in the storage device **125** of the data of the file. Therefore, the block numbers identify the addresses of the file's data blocks on the disk **125**.

An embodiment of the invention includes a block distribution engine **140** and mapping **145**, in order to perform a block distribution algorithm that provides security to data blocks on a storage device **125**. The engine **140** can be programmed by use of standard programming languages (e.g., C, C++, Pascal) and can be programmed by use of standard programming

techniques that are known to those skilled in the art. In previous systems, the data blocks of a file are linearly placed on a storage device and can be obtained by accessing the inode of the file. In contrast, a block distribution algorithm in an embodiment of the invention will vary the distribution of the data blocks on the storage device **125** in order to provide security to the data blocks, as discussed in additional details below.

Reference is now made to FIGS. **1** and **2** for purposes of discussing the operations of embodiments of the invention. A read operation for a file is first discussed and a write operation for the file is then discussed.

During a read operation, if the data blocks for a file **205** (FIG. **2**) are not found by the memory subsystem **130** (FIG. **1**) in the memory cache **120** (FIG. **1**) (in an embodiment that implements the checking of cached data in memory cache **120**), then the disk I/O subsystem **135** sends a query **150** (FIG. **1**) to the block distribution engine **140** which determines if the data blocks of the file **205** is mapped in the mapping **145**. The mapping **145** is a data structure mapping of the blocks in an inode of the file **205** to an algorithm-specific data blocks. Note also that in the example of FIG. **2**, the information on the data blocks (such the locations of the data blocks in the storage device **125**) of a file **205** are identified in the inode **210**. However, it is within the scope of the embodiments of the invention to use other data structures to store the data blocks information of a file **205**, depending on the file system type.

If the data blocks of the file **205** is mapped by mapping **145** to algorithm-specific data blocks, then the block distribution engine **140** returns the algorithm-specific data block numbers **155** (FIG. **1**) to the disk I/O subsystem **135**. These algorithm-specific data block numbers **155** are the mapping of the data block numbers in the inode **210** of the file **205**. An example of the data block number mapping is shown in FIG. **2**. The subsystem **135** then passes the algorithm-specific data block numbers **155** to the block device driver **160** which then reads the data blocks of file **205** at addresses in storage device **125**. The locations of these addresses in the storage device **125** are identified by the algorithm-specific data block numbers **155**. The block device driver **160** can then read the data in the algorithm-specific data block numbers **155** locations in the storage device **125**, and these data are the data of the file **205**.

In contrast, if the data blocks of the file **205** are not mapped by the mapping **145**, then the data block numbers of the inode **210** are passed the block device driver **160**. In this case, the locations of the addresses in the storage device **125** of the data of the file **205** are identified by the data block numbers of the inode **210**. The block device driver **160** can then read the data in the data block numbers of the inode **210**.

An example of the data blocks mapping **145** is discussed with reference to FIG. **2**. Assume that an inode **210** of file **205** points to four (4) data block numbers X**1**-X**4**. Note that a file can have more than four data block numbers in an inode, and X**1**-X**4** are used herein only as an example for purposes of describing the details of an embodiment of the invention. The parameters X**1**-X**4** are locations in the storage device **125**, and X**1**-X**4** will contain the data of the file **205**. The block distribution engine **140** maps the data block numbers X**1**-X**4** to different data block numbers f(X**1**)-f(X**4**), respectively, by applying the equation f(x) to the data block numbers x={X**1**, X**2**, X**3**, X**4**}. The equation f(x) can be any function that is applied to the variables X**1**-X**4**. As an example, assume that the data block numbers for inode **210** are the following: X**1**=1, X**2**=2, X**3**=3, and X**4**=4. Assume also as an example that f(x)=5x, although as noted above, f(x) can be programmed to be other functions. Therefore, the mapped data blocks **215** will be as follows: f(X**1**)=5, f(X**2**)=10, f(X**3**)=15, and f(X**4**)=20 Therefore, the data blocks **215** for the file **205** will be at the locations, data block numbers **5**, **10**, **15**, and **20**, in the storage device **125**. The function f(x) is only an example of an algorithm that the algorithm designer can program for use by the block distribution engine **140**. Therefore, the block distribution engine **140** can apply other algorithms to the data block numbers (x). The algorithm designer has the choice to design different functions f(x) that differ in complexity level, based on, for example, considerations of efficiency and security level of the mapping **145**. For example, functions f(x) that are less complex mathematical equations provide efficiency because the block distribution engine **140** can perform a faster computation for f(x). In contrast, functions f(x) that are more complex mathematical equations provide a higher level of security for the mapping of the data block numbers. The typical characteristics of this function f(x) that the designer can consider should be, for example: (1) every block number (x) that is applied through the function f(x) maps to another unique block number f(x), (2) the mapped block number f(x) maps to only one data block number (x) that was requested by the disk I/O subsystem **135**, (3) the data blocks on storage device **125** (disk) are preferably not wasted (i.e., the distribution of the locations of the address numbers f(x) are efficiently configured on the storage device **125**).

A user who does not have the block distribution engine **140** and mapping **145** installed in his/her computer will not be able to read the data blocks f(X**1**)-f(X**4**) for the file **205** by simply accessing the inode **210** in the storage device **125** due to the mapping function f(x). Therefore, the mapping **145** of the data blocks **212** into the mapped data blocks **215** provides security to the file **205** data that are stored in a storage device **125**. Specifically, the data block numbers X**1**-X**4** (of blocks **212**) are mapped to the algorithm-specific (mapped) data block numbers f(X**1**)-f(X**4**) of the data blocks **215**. An unauthorized user will not be able to obtain the data for a file **205** by simply accessing the inode **210** of the file **205** and copying the data blocks **212**, because the actual data of the file **205** are stored in the mapped data blocks **215**.

During a write operation, the file system **110** assigns an inode (e.g., inode **210**) to a file **205** to be written to the storage device **125**. The disk I/O subsystem **135** identifies the inode number **230** of the inode **210**. From the identified inode number **230**, the disk I/O subsystem **135** identifies the filename **235** of file **205**. The file system has a list of free data blocks (data blocks that presently do not contain data of files). From the list of free data blocks, assume that, e.g., blocks X**1**=1, X**2**=2, X**3**=3 and X**4**=4 are free blocks, and the file system allocates these 4 free blocks X**1**=1, X**2**=2, X**3**=3 and X**4**=4 for storing data of the file **205** to be written to disk. After block allocation and just before writing the data into the data blocks in the storage device **125** (disk), the block distribution engine **140** will apply the mapping function f(x) to all data block numbers X**1**-X**4** that are linked to the filename **235**. This linking will return the corresponding mapped block numbers f(X**1**)=5, f(X**2**)=10, f(X**3**)=15 and f(X**4**)=20 to the disk I/O subsystem **135** when the disk I/O subsystem **135** sends the data block numbers X**1**-X**4** to the block distribution engine **140**. The disk I/O subsystem **135** then sends the mapped block numbers **5**, **10**, **15**, and **20** to the block device driver **160**, and the block device driver **160** (FIG. **1**) will write the data of the file **205** on the data block numbers **5**, **10**, **15**, and **20** on the storage device **125**. The read operation does the reverse operation as explained above and thus the integrity of the file system is not lost because there is linking from the filename **235**, to the allocated data blocks numbers X**1**-X**4** for the file **205**, and then to the mapped data blocks numbers f(X**1**)-f(X**4**). Therefore, the disk I/O subsystem **135** (FIG. **1**)

is able to obtain the correct mapped data block numbers f(X1)-f(X4) for a file **205** to be read, by starting with the filename **235** of the file.

FIG. **3** is a method **300** for a data security to be used in a file system, in accordance with an embodiment of the invention. In block **310**, the VFS processes and forwards the system call with the inode number of a file to the file system. In block **315**, the file system handles the system call. In block **320**, the memory subsystem of the file system handles the system call. In block **325**, if the data blocks of the file are in the memory cache, then the data blocks are returned to the VFS. The steps in blocks **320-325** are optional steps that are performed in embodiments of the invention that implements the checking of cached data in the memory cache **120** (FIG. **1**).

In block **330**, the disk I/O subsystem operates with a block distribution algorithm **335** which performs the following steps. The algorithm **335** determines in block **340** if each data block number in an inode of the file is mapped to an algorithm-specific data block number. An example of the algorithm **335** is the mapping function f(x) of FIG. **2**). In block **340**, if the data block numbers are not mapped to algorithm-specific data block numbers, then the data block numbers in the inode are returned (**345**) to the disk I/O subsystem and these data block numbers **350** are forwarded to a block device driver. In block **355**, the block device driver performs the function of disk access to the storage device **125** based on the data block numbers **350**.

In block **340**, if the data block numbers in the inode of the file are mapped to algorithm-specific data block numbers based on a mapping function f(x), then in block **360**, the block distribution engine **140** (FIG. **1**) converts (maps) the data block numbers X1-X4 (FIG. **2**) to the mapped data block numbers f(X1)-f(X4). In block **365**, this mapping is stored in the data structure mapping **145** of FIG. **1**. The block distribution engine **140** (FIG. **1**) also returns **370** the algorithm-specific (mapped) data block numbers f(X1)-f(X4) to the disk I/O subsystem. In block **355**, the block device driver will access the algorithm-specific data block numbers f(X1)-f(X4) in the storage device **125** in order to obtain the data of a file.

Embodiments of the invention provide various advantages such as providing a frame work for security of data on a storage device, providing various levels of security to the data based on a selected mapping function complexity, providing an opportunity to define security standards at the file system level, and providing a security frame work that can be used with different file system types such as, for example, ext2 (second extended file system) and FAT (File Allocation Table).

It is also within the scope of the present invention to implement a program or code that can be stored in a machine-readable or computer-readable medium to permit a computer to perform any of the inventive techniques described above, or a program or code that can be stored in an article of manufacture that includes a computer readable medium on which computer-readable instructions for carrying out embodiments of the inventive techniques are stored. Other variations and modifications of the above-described embodiments and methods are possible in light of the teaching discussed herein.

The above description of illustrated embodiments of the invention, including what is described in the Abstract, is not intended to be exhaustive or to limit the invention to the precise forms disclosed. While specific embodiments of, and examples for, the invention are described herein for illustrative purposes, various equivalent modifications are possible within the scope of the invention, as those skilled in the relevant art will recognize.

These modifications can be made to the invention in light of the above detailed description. The terms used in the following claims should not be construed to limit the invention to the specific embodiments disclosed in the specification and the claims. Rather, the scope of the invention is to be determined entirely by the following claims, which are to be construed in accordance with established doctrines of claim interpretation.

What is claimed is:

1. A method of providing data security for use with a file system, the method comprising:

applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and

obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.

2. The method of claim **1**, further comprising:

storing a mapping of the data block numbers and the mapped data block numbers.

3. The method of claim **1**, wherein a read operation for the file comprises:

obtaining the data block numbers of the file;

obtaining the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and

reading the data in the mapped data block numbers in the storage device.

4. The method of claim **1**, wherein a write operation for the file comprises:

allocating data block numbers of free data blocks in the storage device;

linking the data block numbers to a file name of the file;

applying the mapping function to the data block numbers in order to obtain the mapped data block numbers; and

writing the data of the file into the mapped data block numbers in the storage device.

5. The method of claim **1**, wherein the mapping function is an equation that is applied to each data block number of the file.

6. The method of claim **1**, further comprising:

selecting a complexity level of the mapping function in order to set a security level and efficiency of the mapping function.

7. The method of claim **1**, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.

8. The method of claim **1**, wherein the mapping function prevents copying of data of the file in the storage device.

9. An apparatus for providing data security for use with a file system, the apparatus comprising:

a block distribution engine configured to apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file, and obtain mapped data block numbers after applying the mapping function;

wherein the mapped data block numbers are addresses of data of the file in a storage device.

10. The apparatus of claim **9**, wherein the block distribution engine is configured to store a mapping of the data block numbers and the mapped data block numbers.

11. The apparatus of claim **9**, wherein for a read operation for the file, the block distribution engine obtains the data

block numbers of the file from a file system, and obtains the mapped data block numbers of the file based on the mapping function that is applied to the data block numbers; and wherein a block device driver is configured to read the data in the mapped data block numbers in the storage device.

12. The apparatus of claim **9**, wherein for a write operation for the file, the block distribution engine is configured to receive allocated data block numbers of free data blocks in the storage device from a file system, link the data block numbers to a file name of the file, and apply the mapping function to the data block numbers in order to obtain the mapped data block numbers; and wherein a block device driver is configured to write the data of the file into the mapped data block numbers in the storage device.

13. The apparatus of claim **9**, wherein the mapping function is an equation that is applied to each data block number of the file.

14. The apparatus of claim **9**, wherein a complexity level of the mapping function is selected in order to set a security level and efficiency of the mapping function.

15. The apparatus of claim **9**, wherein the data block numbers are addresses of data of the file in the storage device if the mapping function is not applied to the data block numbers.

16. The apparatus of claim **9**, wherein the mapping function prevents copying of data of the file in the storage device.

17. An apparatus for providing data security for use with a file system, the apparatus comprising:

means for applying a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and

means for obtaining mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.

18. Article of manufacture, comprising:

a machine-readable medium having stored thereon instructions to:

apply a mapping function to data block numbers that are associated with a file, wherein the data block numbers are contained in an index node associated with said file; and

obtain mapped data block numbers after applying the mapping function, wherein the mapped data block numbers are addresses of data of the file in a storage device.

* * * * *

EXHIBIT C

US007640332B2

(12) **United States Patent**
Dahiya

(10) **Patent No.: US 7,640,332 B2**
(45) **Date of Patent: Dec. 29, 2009**

(54) **SYSTEM AND METHOD FOR HOT DEPLOYMENT/REDEPLOYMENT IN GRID COMPUTING ENVIRONMENT**

(75) Inventor: **Sanjay Dahiya**, Karnataka (IN)

(73) Assignee: **Hewlett-Packard Development Company, L.P.**, Houston, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 256 days.

(21) Appl. No.: **12/003,182**

(22) Filed: **Dec. 20, 2007**

(65) **Prior Publication Data**

US 2008/0162674 A1 Jul. 3, 2008

(30) **Foreign Application Priority Data**

Dec. 27, 2006 (IN) 2428/CHE/2006

(51) **Int. Cl.**
G06F 15/173 (2006.01)
G06F 9/445 (2006.01)

(52) **U.S. Cl.** **709/223**; 717/176

(58) **Field of Classification Search** 709/223; 717/176

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,305,585	B2 *	12/2007	Marchand	714/18
2005/0120353	A1 *	6/2005	Zhang et al.	719/310
2005/0138618	A1 *	6/2005	Gebhart	717/176
2006/0005181	A1 *	1/2006	Fellenstein et al.	717/174
2006/0195688	A1 *	8/2006	Drissi et al.	713/152
2007/0028098	A1 *	2/2007	Baartman et al.	713/162
2007/0106773	A1 *	5/2007	Gallino et al.	709/223
2007/0233827	A1 *	10/2007	McKnight	709/223
2008/0123559	A1 *	5/2008	Haviv et al.	370/255

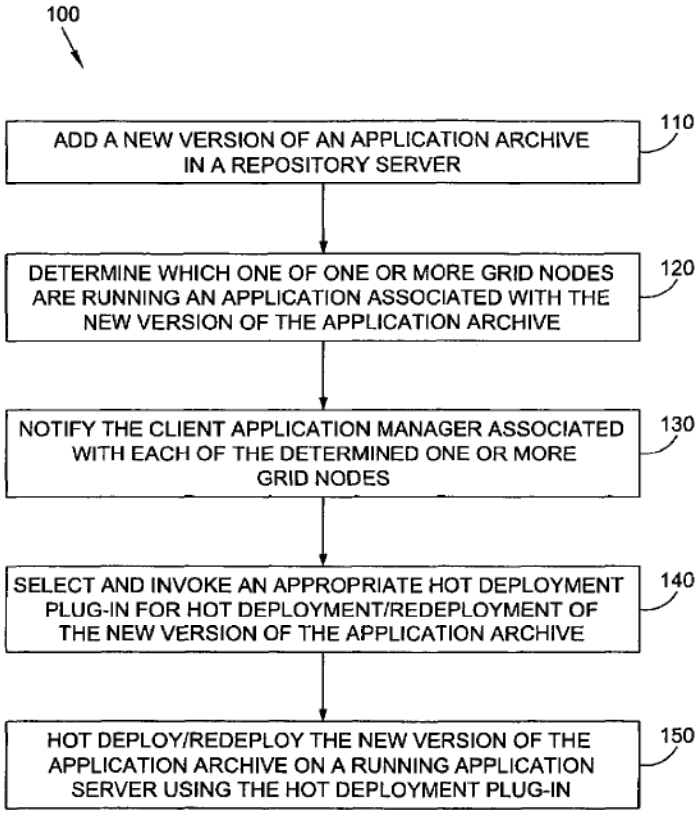
* cited by examiner

Primary Examiner—Kenny S Lin

(57) **ABSTRACT**

A technique for hot deploying/redeploying applications in a grid computing environment to improve operating efficiency and reduced overhead. In one example embodiment, this is accomplished by notifying a client application manager associated with each of one or more grid nodes about a type of data transfer protocol to use upon receiving a new version of an application release bundle by a repository server. The new version of the application release bundle is then hot deployed/redeployed on running one or more application servers in the associated grid using an appropriate hot deployment plug-in based on the data transfer protocol by each client application manager.

18 Claims, 3 Drawing Sheets



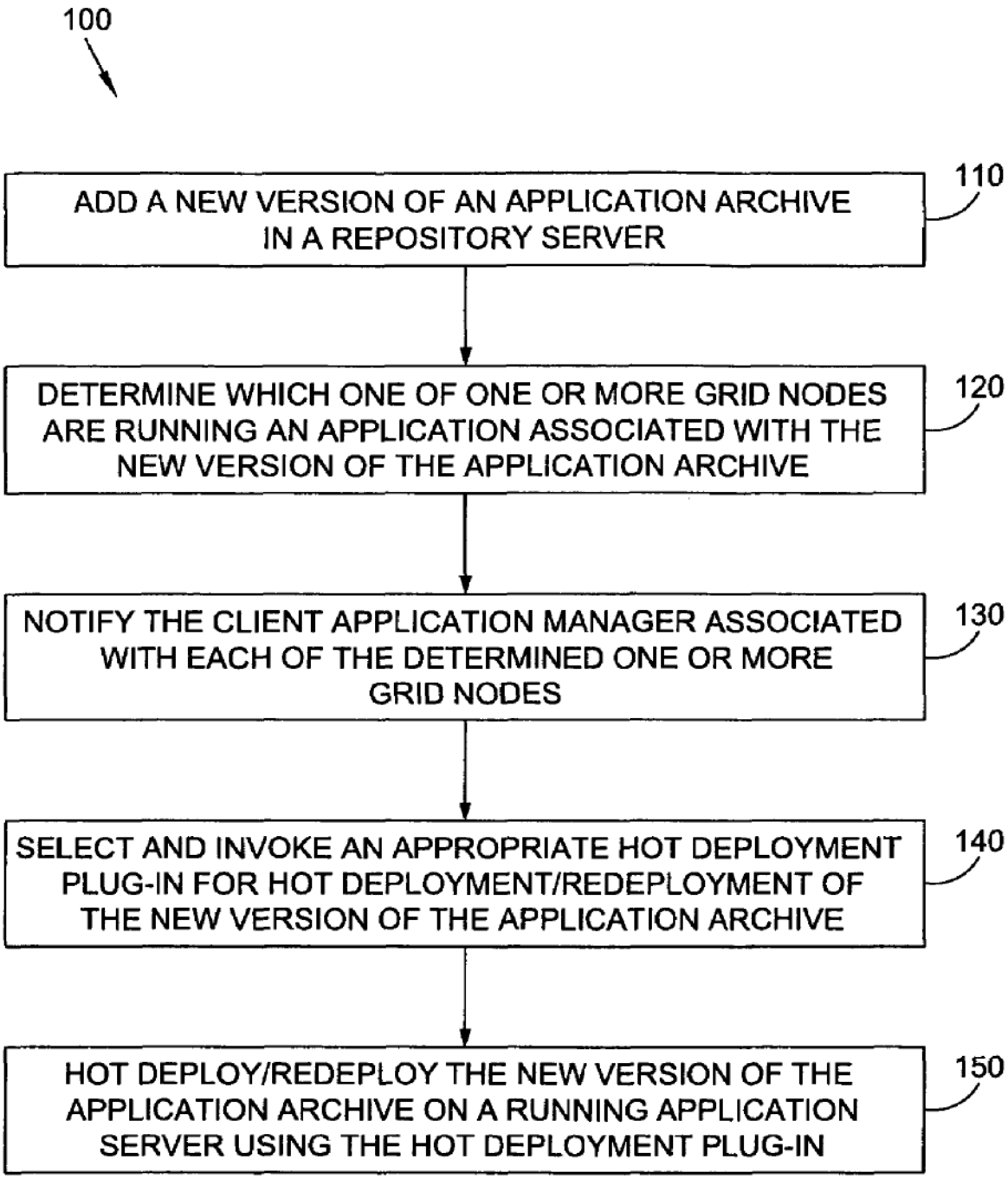

100
ADD A NEW VERSION OF AN APPLICATION ARCHIVE IN A REPOSITORY SERVER
110
DETERMINE WHICH ONE OF ONE OR MORE GRID NODES ARE RUNNING AN APPLICATION ASSOCIATED WITH THE NEW VERSION OF THE APPLICATION ARCHIVE
120
NOTIFY THE CLIENT APPLICATION MANAGER ASSOCIATED WITH EACH OF THE DETERMINED ONE OR MORE GRID NODES
130
SELECT AND INVOKE AN APPROPRIATE HOT DEPLOYMENT PLUG-IN FOR HOT DEPLOYMENT/REDEPLOYMENT OF THE NEW VERSION OF THE APPLICATION ARCHIVE
140
HOT DEPLOY/REDEPLOY THE NEW VERSION OF THE APPLICATION ARCHIVE ON A RUNNING APPLICATION SERVER USING THE HOT DEPLOYMENT PLUG-IN
150

FIG. 1

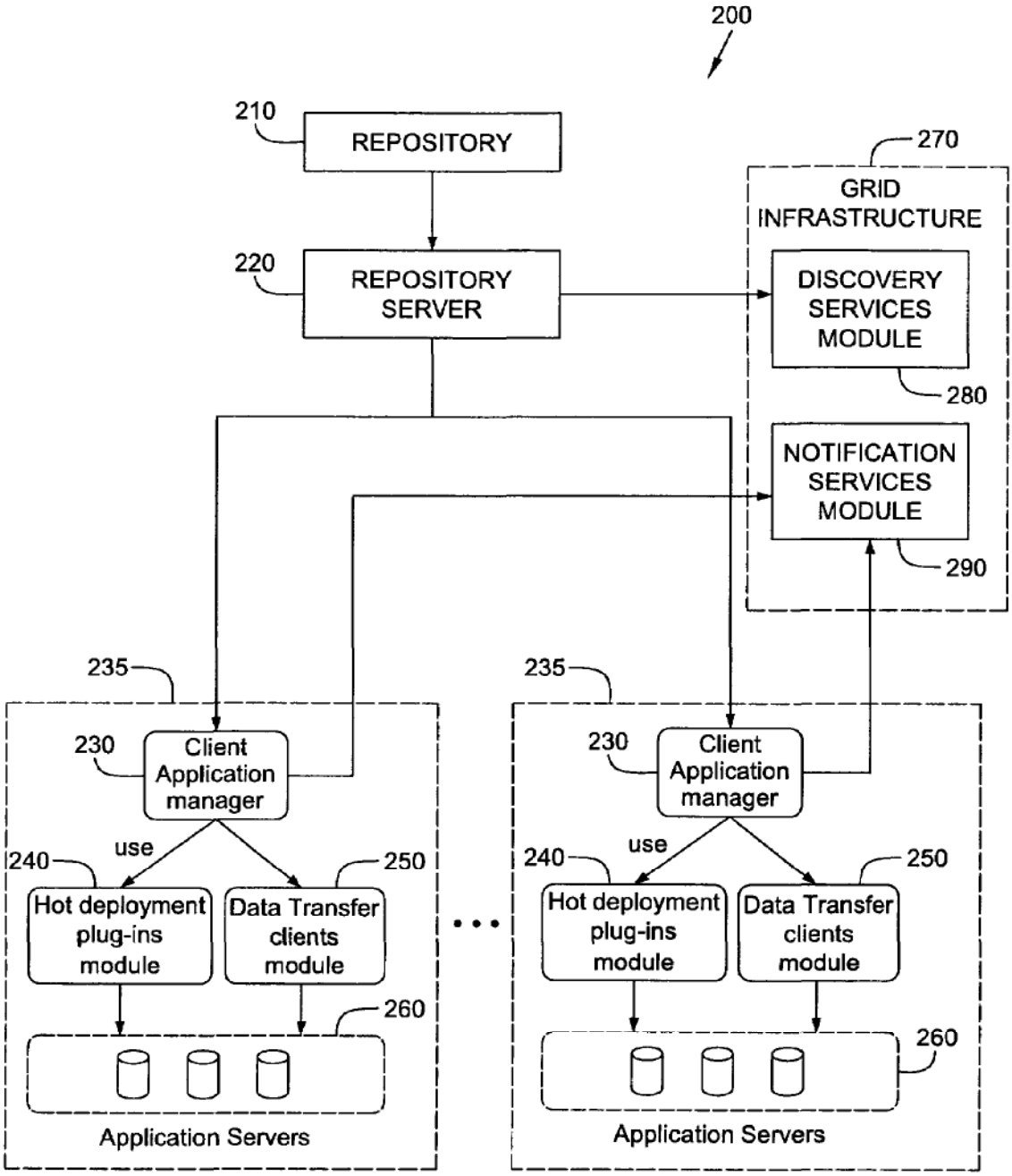

200
210
REPOSITORY
220
REPOSITORY SERVER
270
GRID INFRASTRUCTURE
DISCOVERY SERVICES MODULE
280
NOTIFICATION SERVICES MODULE
290
235
230
Client Application manager
use
240
Hot deployment plug-ins module
250
Data Transfer clients module
260
Application Servers
235
230
Client Application manager
use
240
Hot deployment plug-ins module
250
Data Transfer clients module
260
Application Servers

FIG. 2

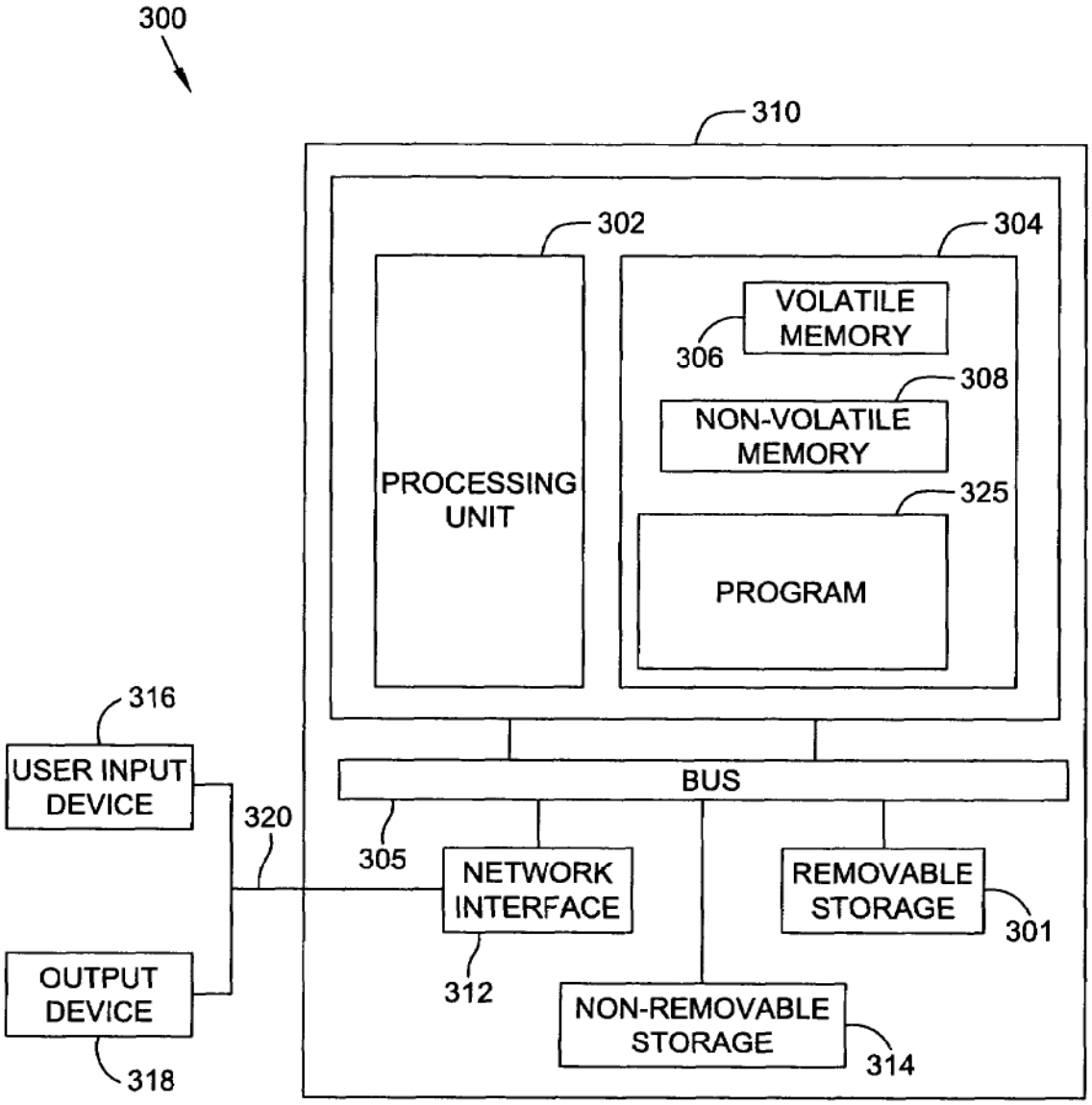

300
310
302
304
VOLATILE MEMORY
306
308
NON-VOLATILE MEMORY
PROCESSING UNIT
325
PROGRAM
316
USER INPUT DEVICE
320
BUS
305
NETWORK INTERFACE
REMOVABLE STORAGE
301
OUTPUT DEVICE
312
NON-REMOVABLE STORAGE
314
318

FIG. 3

SYSTEM AND METHOD FOR HOT DEPLOYMENT/REDEPLOYMENT IN GRID COMPUTING ENVIRONMENT

This application claims priority from Indian patent application 2428/CHE/2006, filed on Dec. 27, 2006. The entire content of the aforementioned application is incorporated herein by reference.

BACKGROUND OF THE INVENTION

Grid computing is touted as one of the most important and promising paradigms for applications that require long computing times or large storage spaces. It is attracting a growing number of users to develop larger scale distributed software applications than ever before. In general, grid computing enables resource virtualization, on-demand provisioning, and service (resource) sharing between organizations. In summary, grid computing provides the ability to use a set of open standards and protocols, to gain access to applications and data, processing power, storage capacity and a vast array of other computing resources over the Internet. It is a type of parallel and distributed system that enables the sharing, selection, and aggregation of resources distributed across multiple administrative domains based on the resources availability, capacity, performance, cost and users' quality-of-service requirements.

In recent years, grid computing with web services for components oriented systems, such as J2EE based applications is gaining popularity. The J2EE based applications are built as a set of components. Generally, these components are hosted by a container or a server, which provides an execution environment to the components. In such a system, multiple applications may be hosted in a single container. In such a scenario, upgrading an application or hot redeployment of an application in a container does not require restarting the container as J2EE based container support hot deployment mechanisms. This is a very desirable feature as it would not impact running of other applications residing in the system. Hot deployment refers to a process of deploying/redeploying an application without having to shutdown/restart an application container. Hot deployment is commonly used in containers that run in standalone mode or homogenous cluster environment where every new application change is propagated to all the servers connected in the environment.

Whereas, in grid computing environments there can be potentially many containers running several applications and different servers can be running different set of applications, i.e., the applications can run in multi-vendor and multi-version application server environments. Such a complexity can create a two fold problem in grid computing environment. The first being that the information about which server is running what application is dynamic and not static. The second being that the application containers may be running on different platforms as described above and can require different data transfer mechanisms for hot deployment/redeployment. This can result in requiring the grid system to be temporarily suspended and reinitialized during the service deployment/redeployment. This can further result in requiring stopping all the running services and applications. This can be a significant overhead on the grid system.

BRIEF DESCRIPTION OF THE DRAWINGS

Embodiments of the invention will now be described, by way of example only, with reference to the accompanying drawings in which:

FIG. **1** is a flowchart illustrating a method for hot deployment/redeployment in grid computing environment according to an embodiment of the present subject matter.

FIG. **2** is a block diagram that shows how the hot deployment system interacts in a grid computing environment according to an embodiment of the present subject matter.

FIG. **3** is a block diagram of a typical computer system used for implementing embodiments of the present subject matter shown in FIGS. **1-2**.

DETAILED DESCRIPTION OF THE INVENTION

In the following detailed description of the various embodiments of the invention, reference is made to the accompanying drawings that form a part hereof, and in which are shown by way of illustration specific embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention, and it is to be understood that other embodiments may be utilized and that changes may be made without departing from the scope of the present invention. The following detailed description is, therefore, not to be taken in a limiting sense, and the scope of the present invention is defined only by the appended claims.

The terms "application server" and "application container" are used interchangeably throughout the document.

The present technique tightly integrates deployment and/or re-deployment system with grid discovery services. Further, the technique provides data transfer services to find the deployed and/or redeployed instances of target applications, move new applications on target nodes and then hot-deploy and re-deploy them.

FIG. **1** illustrates an example method **100** for hot deployment/redeployment of data in grid computing environment. At step **110**, this example method **100** begins by adding a new version of the application release bundle in a repository server. In these embodiments, the repository server contains application release bundles for all versions of an application. These application release bundles are available to all nodes on the grid system. Also in these embodiments, the repository server can support one or more data transfer protocols, such as ftp (file transfer protocol), http (hypertext transfer protocol), gridftp, tcp (transmission control protocol), udp (user datagram protocol) and so on to transfer application release bundles to the associated client nodes. In some embodiments, the grid computing environment can include multiple grid nodes and each node can have hundreds of servers running on heterogeneous platforms. These servers generally run different set of applications at different times. The repository server is kept updated so that the next time when an application is deployed an updated version is chosen from the repository server thereby significantly reducing the chance of deploying an older version of the application. The application release bundle can include application archive code, such as zip, jar, tar, tgz, exe and the like.

At step **120**, upon adding the new version of the application release bundle in the repository server, the method **100** determines which one of the one or more grid nodes are running an application associated with the added new version of the application release bundle by a discovery services module. At step **130**, a client application manager associated with each of the determined one or more grid nodes notifies the adding of the new version of the application release bundle along with the type of data transfer protocol to use by the repository server. The term client refers to nodes running application servers on each grid node. The grid nodes include one or more application servers that include components, such as Web

containers, Web service containers, and EJB container. In these embodiments, the client application manager registers with the repository server and can accept notifications from a server using grid notification mechanisms. The client application manager uses data transfer plug-ins for transferring application release bundles from the repository server to client and hence is extensible to support any data transfer mechanism. In some embodiments, the client application manager associated with each one of the one or more grid nodes registers with the repository server for accepting notification of receiving the new version of the application release bundle.

At step **140**, an appropriate hot-deployment plug-in is selected and invoked by each client application manager based on the data transfer protocol to use for hot deploying/redeploying the new version of the application release bundle in an associated grid node. In these embodiments, the application includes a set of components that are hosted by one or more application servers in the one or more grid nodes.

At step **150**, the new version of the application release bundle is hot deployed/redeployed on running one or more application servers in the associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by each client application manager. In some embodiments, the new version of the application release bundle is hot deployed/redeployed on running one or more application servers using the appropriate hot deployment plug-in by a data transfer client associated with each client application manager. In these embodiments, the repository server includes one or more new versions of an application release bundle associated with the application.

Although the flowchart **100** includes steps **110-150** that are arranged serially in the exemplary embodiments, other embodiments of the subject matter may execute two or more steps in parallel, using multiple processors or a single processor organized as two or more virtual machines or sub-processors. Moreover, still other embodiments may implement the steps as two or more specific interconnected hardware modules with related control and data signals communicated between and through the modules, or as portions of an application-specific integrated circuit. Thus, the exemplary process flow diagrams are applicable to software, firmware, and/or hardware implementations.

Referring now to FIG. **2**, there is illustrated an example block diagram of a hot deployment system **200** and its interactions within a grid computing environment. As shown in FIG. **2**, the hot deployment system **200** includes a repository **210**, a repository server **220**, one or more grid nodes **235** and a grid infrastructure **270**. Further as shown in FIG. **2**, each grid node **235** includes a client application manager **230**, hot deployment plug-ins **240** and data transfer clients **250**, and one or more application servers **260** that are associated with each client application manager **230**. In these embodiments, one or more deployed applications run on the one or more application servers **260** in each grid node **235**. The one or more application servers **260** include components, such as Web containers, Web service container, and EJB container. Furthermore as shown in FIG. **2**, the grid infrastructure **270** includes discovery services module **280** and notification services module **290**.

In operation, the repository **210** adds a new version of an application release bundle in the repository server **220**. Generally in these embodiments, the repository server **220** contains application release bundles for all versions of an application. These application release bundles are available to all grid nodes **235** in the grid computing environment. In these embodiments, the repository server **220** can support one or more data transfer protocols. Exemplary data transfer protocols include ftp, http, gridftp, tcp, and udp.

Upon adding the new version of the application release bundle in the repository server **220** by the repository **210**, the discovery services module **280** determines which of the one or more grid nodes **235** are running an application associated with the added new version of the application release bundle. The repository server **220** then notifies the client application manager **230** associated with the determined one or more grid nodes **235** about adding the new version of the application release bundle along with the type of data transfer protocol to use for hot deploying/redeploying the new version of the application release bundle. The term client refers to a node running the one or more application servers **260** in each grid node. In these embodiments, each client application manager **230** registers with the repository server **230** and can accept the notification from the repository server **230** about the adding of a new version of an application release bundle in the repository server **230** via the grid infrastructure **270**. The discovery services module **280** includes information related to resources in each grid node **235**. The resource information can include protocols, such as G I I S (Grid Information Index Service) and G R I S (Grid Resource Information Service), required to hot deploy/redeploy the new version of the application release bundle for each grid node **235**. The discovery services module **280** also includes information relating to each type of data transfer protocol, for example, ftp, http, gridftp, tcp, and udp, required for hot deploying/redeploying. The notification services module **290** includes scheduling information, for example, Meta data relating to scheduling information. In these embodiments, each client application manager **230** obtains the information associated with added new version of the application release bundle and hot deploys/redeploys the new version of the application release bundle in the associated one or more application servers **260** in the associated grid node **235**.

In some embodiments, each client application manager **230** hot deploys/redeploys the new version of the application release bundle on one or more running application servers **260** in associated grid node **235** using an appropriate hot deployment plug-in based on using the notified type of data transfer protocol for hot deploying/redeploying. In some embodiments, the client application manager **230** associated with each grid node **235** selects an appropriate hot deployment plug-in needed to hot deploy/redeploy the new version of the application release bundle in the associated one of the one or more application servers **260** using the grid infrastructure **270**. In these embodiments, the client application manager **230** selects an appropriate hot deployment plug-in using the discovery services module **280** and hot deploys/redeploys the new version of the application release bundle via the hot deployment plug-ins module **240** on one or more running application servers **260** in each determined grid node **235**. The application release bundle includes, for example, a set of components that are hosted by one or more application servers **260** in the one or more grid nodes **235**.

The client application manager **230** then invokes the hot deployment plug-ins module **240** along with the data transfer clients module **250** to hot deploy/redeploy the new version of the application release bundle in the associated one or more applications servers **260** in each grid node **235**. In these embodiments, the client application manager **230** interacts with the discovery services module **280** to determine which of the one or more grid nodes are running the application associated with the added new version of the application release bundle. The client application manager **230** associated with each grid node **235** hot deploys/redeploys the new version of

the application release bundle in the one or more application servers **260** using the hot deployment plug-ins module **240** and the data transfer clients module **250**.

The hot deployment/redeployment plug-ins are selected by each client application manager **230** based on system hardware information associated with each of the one or more application servers **260** obtained from the grid infrastructure **270**. The hot deployment/redeployment plug-ins can include protocols, such as ftp, http, gridftp, tcp, and udp. These hot deployment/redeployment plug-ins facilitate in hot deploying/redeploying the added new version of the application release bundle on running one or more applications servers **260**. In these embodiments, the hot deployment/redeployment plug-ins are used for data transfer associated with the added new version of the application release bundle. Also in these embodiments, the client application manager **230** uses the data transfer clients module **250** via the hot deployment plug-ins module **240** to transfer data associated with added new version of the application release bundle.

Various embodiments of the present subject matter can be implemented in software, which may be run in the computing system environment **300** shown in FIG. **3** (to be described below) or in any other suitable computing environment. The embodiments of the present subject matter are operable in a number of general-purpose or special-purpose computing environments. Some computing environments include personal computers, general-purpose computers, server computers, hand-held devices (including, but not limited to, telephones and personal digital assistants (PDAs) of all types), laptop devices, multi-processors, microprocessors, set-top boxes, programmable consumer electronics, network computers, minicomputers, mainframe computers, distributed computing environments and the like to execute code stored on a computer-readable medium. The embodiments of the present subject matter may be implemented in part or in whole as machine-executable instructions, such as program modules that are executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, and the like to perform particular tasks or to implement particular abstract data types. In a distributed computing environment, program modules may be located in local or remote storage devices.

FIG. **3** shows an example of a suitable computing system environment **300** for implementing embodiments of the present subject matter. FIG. **3** and the following discussion are intended to provide a brief, general description of a suitable computing environment in which certain embodiments of the inventive concepts contained herein may be implemented.

A general computing device, in the form of a computer **310**, may include a processor **302**, memory **304**, removable storage **301**, and non-removable storage **314**. Computer **310** additionally includes a bus **305** and a network interface **312**.

Computer **310** may include or have access to a computing environment that includes one or more user input devices **316**, one or more output devices **318**, and one or more communication connections **320** such as a network interface card or a USB connection. The one or more output devices **318** can be a display device of computer, computer monitor, TV screen, plasma display, LCD display, display on a digitizer, display on an electronic tablet, and the like. The computer **310** may operate in a networked environment using the communication connection **320** to connect to one or more remote computers. A remote computer may include a personal computer, server, router, network PC, a peer device or other network node, and/or the like. The communication connection may include a Local Area Network (LAN), a Wide Area Network (WAN), and/or other networks.

The memory **304** may include volatile memory **306** and non-volatile memory **308**. A variety of computer-readable media may be stored in and accessed from the memory elements of computer **310**, such as volatile memory **306** and non-volatile memory **308**, removable storage **301** and non-removable storage **314**. Computer memory elements can include any suitable memory device(s) for storing data and machine-readable instructions, such as read only memory (ROM), random access memory (RAM), erasable programmable read only memory (EPROM), electrically erasable programmable read only memory (EEPROM), hard drive, removable media drive for handling compact disks (CDs), digital video disks (DVDs), diskettes, magnetic tape cartridges, memory cards, Memory Sticks™, and the like; chemical storage; biological storage; and other types of data storage.

"Processor" or "processing unit," as used herein, means any type of computational circuit, such as, but not limited to, a microprocessor, a microcontroller, a complex instruction set computing (CISC) microprocessor, a reduced instruction set computing (RISC) microprocessor, a very long instruction word (VLIW) microprocessor, explicitly parallel instruction computing (EPIC) microprocessor, a graphics processor, a digital signal processor, or any other type of processor or processing circuit. The term also includes embedded controllers, such as generic or programmable logic devices or arrays, application specific integrated circuits, single-chip computers, smart cards, and the like.

Embodiments of the present subject matter may be implemented in conjunction with program modules, including functions, procedures, data structures, application programs, etc., for performing tasks, or defining abstract data types or low-level hardware contexts.

Machine-readable instructions stored on any of the above-mentioned storage media are executable by the processing unit **302** of the computer **310**. For example, a program module **325** may include machine-readable instructions capable of hot deploying/redeploying data in a grid computing environment according to the teachings and herein described embodiments of the present subject matter. In one embodiment, the program module **325** may be included on a CD-ROM and loaded from the CD-ROM to a hard drive in non-volatile memory **308**. The machine-readable instructions cause the computer **310** to encode according to the various embodiments of the present subject matter.

The operation of the computer system **300** for recompiling hot deploying/redeploying data in a grid computing environment is explained in more detail with reference to FIGS. **1** and **2**.

The above-described technique allows the distributed applications with diverse application containers in a grid environment to be hot deployed and/or redeployed without having to temporarily suspend and/or reinitialize to reduce overhead on the grid system.

The above technique can be implemented using an apparatus controlled by a processor where the processor is provided with instructions in the form of a computer program constituting an aspect of the above technique. Such a computer program may be stored in storage medium as computer readable instructions so that the storage medium constitutes a further aspect of the present subject matter.

The above description is intended to be illustrative, and not restrictive. Many other embodiments will be apparent to those skilled in the art. The scope of the subject matter should

therefore be determined by the appended claims, along with the full scope of equivalents to which such claims are entitled.

As shown herein, the present subject matter can be implemented in a number of different embodiments, including various methods, a circuit, an I/O device, a system, and an article comprising a machine-accessible medium having associated instructions.

Other embodiments will be readily apparent to those of ordinary skill in the art. The elements, algorithms, and sequence of operations can all be varied to suit particular requirements. The operations described-above with respect to the method illustrated in FIG. **1** can be performed in a different order from those shown and described herein.

FIGS. **1-3** are merely representational and are not drawn to scale. Certain proportions thereof may be exaggerated, while others may be minimized. FIGS. **1-3** illustrate various embodiments of the subject matter that can be understood and appropriately carried out by those of ordinary skill in the art.

In the foregoing detailed description of the embodiments of the invention, various features are grouped together in a single embodiment for the purpose of streamlining the disclosure. This method of disclosure is not to be interpreted as reflecting an intention that the claimed embodiments of the invention require more features than are expressly recited in each claim. Rather, as the following claims reflect, inventive invention lies in less than all features of a single disclosed embodiment. Thus, the following claims are hereby incorporated into the detailed description of the embodiments of the invention, with each claim standing on its own as a separate preferred embodiment.

The invention claimed is:

1. A method for hot deployment and/or redeployment in grid computing environment, wherein the grid computing environment includes one or more grid nodes, comprising:

adding a new version of an application release bundle in a repository server;

determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server;

notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use; and

hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers.

2. The method of claim **1**, wherein hot deploying/redeploying the new version of the application release bundle on the running one or more application servers in the associated grid using the appropriate hot deployment plug-in based on the data transfer protocol by the respective client application manager, comprises:

selecting and invoking an appropriate hot deployment plug-in by a respective one of the client application managers based on the data transfer protocol to use for hot deploying/redeploying the new version of the application release bundle in an associated grid node; and

hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in in the associated grid node by the respective client application manager.

3. The method of claim **1**, wherein hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by the respective client application manager comprises:

hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by a data transfer client associated with a respective one of the client application managers.

4. The method of claim **1**, wherein the application comprises a set of components that are hosted by one or more application servers in the one or more grid nodes.

5. The method of claim **1**, wherein the repository sewer stores one or more new versions of the application release bundle associated with the application.

6. The method of claim **1**, further comprising:

registering the client application manager associated with one or more of the determined grid nodes with the repository server for accepting notification of receipt of the new version of the application release bundle.

7. The method of claim **1**, wherein the data transfer plug-in includes protocols selected from the group consisting of ftp, http, gridftp, tcp, and udp.

8. The method of claim **1**, wherein each grid node comprises one or more application servers that include components selected from the group consisting of Web containers, Web service containers, and EJB containers.

9. A computer-readable medium with program code embodied therein for causing when executed, a computer system, to perform

a method for hot deployment/redeployment in a grid computing environment comprising:

adding a new version of an application release bundle in a repository server;

determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server;

notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use; and

hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers.

10. The computer-readable medium of claim **9**, wherein hot deploying/redeploying the new version of the application release bundle on the running one or more application servers in the associated grid using the appropriate hot deployment plug-in based on the data transfer protocol by the respective client application manager, comprises:

selecting and invoking an appropriate hot deployment plug-in by a respective one of the client application managers based on the data transfer protocol to use for hot deploying/redeploying the new version of the application release bundle in an associated grid node; and

hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in in the associated grid node by the respective client application manager.

11. The computer-readable medium of claim 9, wherein hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by the respective client application manager comprises:

hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by a data transfer client associated with a respective one of the client application managers.

12. The computer-readable medium of claim 9, wherein the method further comprises:

registering the client application manager associated with one or more of the determined grid nodes with the repository server for accepting notification of receipt of the new version of the application release bundle.

13. The computer-readable medium of claim 9, wherein the data transfer plug-in includes protocols selected from the group consisting of ftp, http, gridftp, tcp, and udp.

14. A computer system comprising:

a processing unit; and

a memory coupled to the processing unit, the memory having stored therein code for causing, when executed, the processing unit to perform a method comprising:

adding a new version of an application release bundle in a repository server;

determining by a discovery services module which of the one or more grid nodes are running an application associated with the added new version of the application release bundle upon adding the new version of the application release bundle in the repository server;

notifying a client application manager associated with one or more of the determined grid nodes about adding the new version of the application release bundle along with a type of data transfer protocol to use; and

hot deploying/redeploying the new version of the application release bundle on running one or more application servers in an associated grid node using an appropriate hot deployment plug-in based on the data transfer protocol by a respective one of the client application managers.

15. The system of claim 14, wherein hot deploying/redeploying the new version of the application release bundle on the running one or more application servers in the associated grid using the appropriate hot deployment plug-in based on the data transfer protocol by the respective client application manager, comprises:

selecting and invoking an appropriate hot deployment plug-in by a respective one of the client application managers based on the data transfer protocol to use for hot deploying/redeploying the new version of the application release bundle in an associated grid node; and

hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in in the associated grid node by the respective client application manager.

16. The system of claim 14, wherein hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by the respective client application manager comprises:

hot deploying/redeploying the new version of the application release bundle on running one or more application servers using the appropriate hot deployment plug-in by a data transfer client associated with a respective one of the client application managers.

17. The system of claim 14, wherein the method further comprises:

registering the client application manager associated with one or more of the determined grid nodes with the repository server for accepting notification of receipt of the new version of the application release bundle.

18. The system of claim 14, wherein the application comprises a set of components that are hosted by one or more application servers in the one or more grid nodes.

* * * * *

EXHIBIT D

US007120832B2

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

(10) **Patent No.: US 7,120,832 B2**
(45) **Date of Patent: Oct. 10, 2006**

(54) **STORAGE DEVICE PERFORMANCE MONITOR**

(75) Inventors: **Kevin Collins**, Fort Collins, CO (US); **Michael P. Fleischmann**, Fort Collins, CO (US)

(73) Assignee: **Hewlett-Packard Development Company, L.P.**, Houston, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 658 days.

(21) Appl. No.: **09/966,953**

(22) Filed: **Sep. 27, 2001**

(65) **Prior Publication Data**

US 2003/0061546 A1 Mar. 27, 2003

(51) **Int. Cl.**
G06F 11/00 (2006.01)

(52) **U.S. Cl.** **714/42**; 714/718

(58) **Field of Classification Search** 714/42, 714/47, 54, 5–6, 718, 769, 770; 711/204
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,506,955	A	4/1996	Chen et al.	395/183.02
5,566,317	A	10/1996	Treiber et al.	395/444
5,684,945	A	11/1997	Chen et al.	395/182.18
5,802,069	A *	9/1998	Coulson	714/718
5,859,823	A *	1/1999	Yamamuro	369/47.14
5,956,750	A *	9/1999	Yamamoto et al.	711/167
6,101,619	A *	8/2000	Shin	714/710
6,223,252	B1 *	4/2001	Bandera et al.	711/114
6,249,887	B1 *	6/2001	Gray et al.	714/47
6,289,484	B1 *	9/2001	Rothberg et al.	714/769
6,327,106	B1 *	12/2001	Rothberg	360/53
6,401,214	B1 *	6/2002	Li	714/6
6,408,406	B1 *	6/2002	Parris	714/41
6,412,089	B1 *	6/2002	Lenny et al.	714/769
6,460,151	B1 *	10/2002	Warwick et al.	714/718
6,571,354	B1 *	5/2003	Parks et al.	714/7
2002/0184580	A1 *	12/2002	Archibald et al.	714/719

OTHER PUBLICATIONS

"Making Sure Disk Performance Statistics are Collected", Microsoft Corporation, 2000, 2 pages.
"Monitoring Hard Disk Performance" i386, Dublin, Ireland, 2001, 2 pages.
"Using th Performance Monitor", Microsoft Corporation, 2000, 2 pages.
"Pre-Failure Alerting", Dell Computer Corporation, published to the Internet at least as early as Apr. 9, 2001, 1 page.
"Efficient, Real-Time Management of Storage Resources", U.S. Appl. No. 09/589,144, filed Jun. 7, 2000, Duggan, et al.
"Hard Disk Drives", IBM Corporation, published to the Internet at least as early as Sep. 17, 2001, 5 pages.
"S.M.A.R.T., Advanced Drive Diagnostics", Quantum Corporation, 1998, 13 pages. NOTE: This article was printed from a local cache directory and therefore, some figures are unavailable.
"Information Brief: Self-Monitoring, Analysis and Reporting Technology", IBM Group, 1998, 4 pages.
"Playing it Smart", Eric Ottem, Judy Plummer, Seagate Technology White Papers, Jun. 1995, 4 pages.
"Data Lifeguard Online Diagnostics S.M.A.R.T. Test", Western Digital Corporation, published to the Internet at least as early as Sep. 28, 2000, 2 pages.

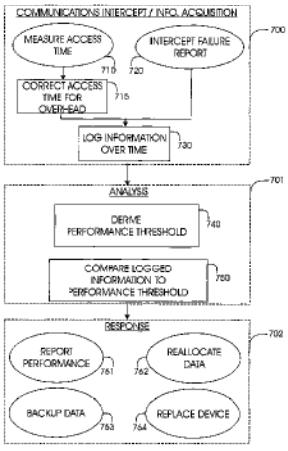
* cited by examiner

Primary Examiner—Dieu-Minh Le

(57) **ABSTRACT**

An apparatus and method for monitoring performance of a storage device. Preferably, the apparatus is embodied in computer readable program code. The apparatus and method may intercept communications between a computer system and the storage device, wherein the communications are compared to a threshold value and/or a predicted failure of the storage device. The apparatus and method may also respond to a decline in the performance of the storage device based on the analyzed communications, and preferably prior to the predicted failure thereof.

26 Claims, 7 Drawing Sheets



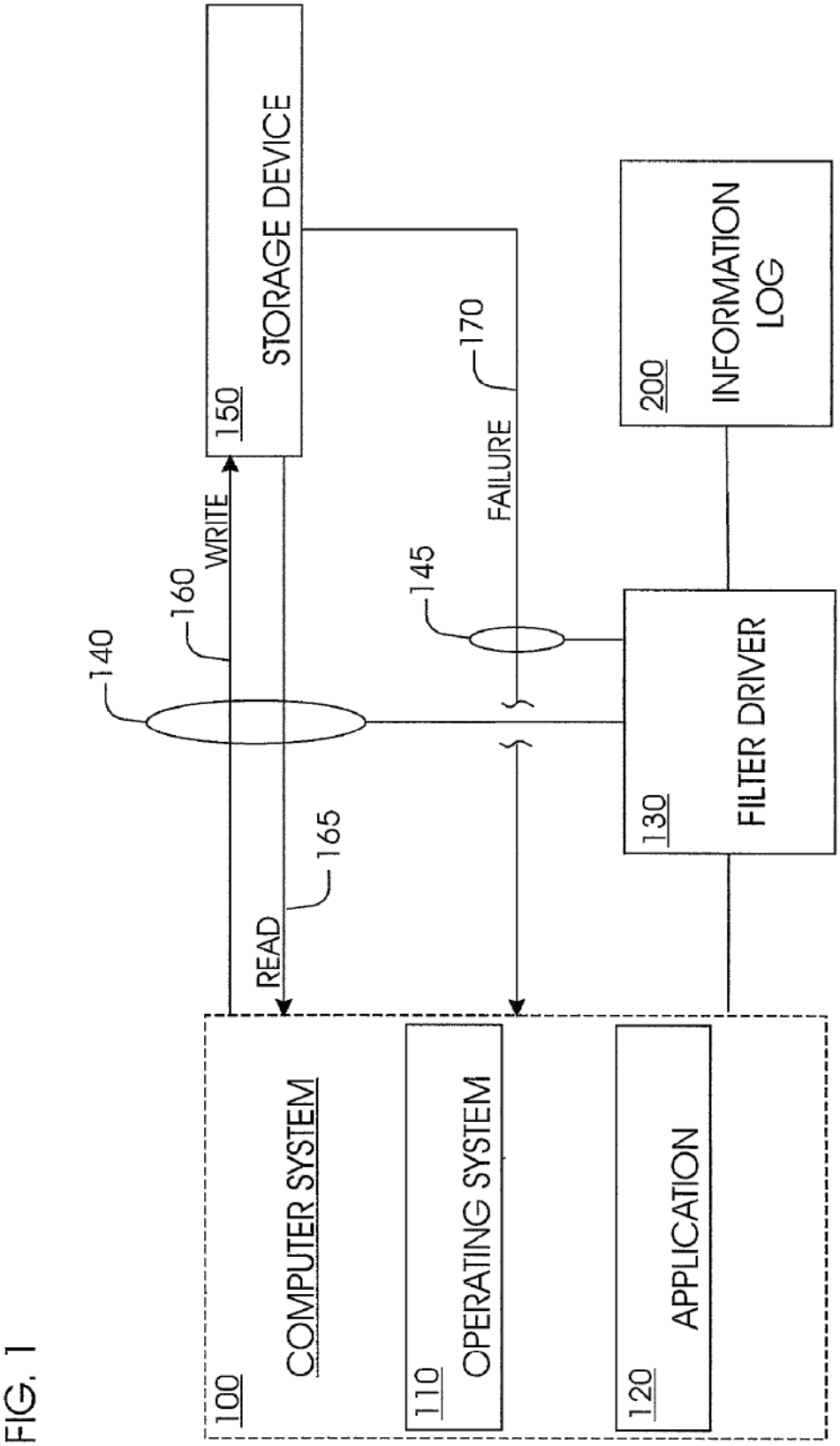

STORAGE DEVICE
150
INFORMATION LOG
200
170
FAILURE
WRITE
160
145
140
FILTER DRIVER
130
165
READ
COMPUTER SYSTEM
100
OPERATING SYSTEM
110
APPLICATION
120

FIG. 1

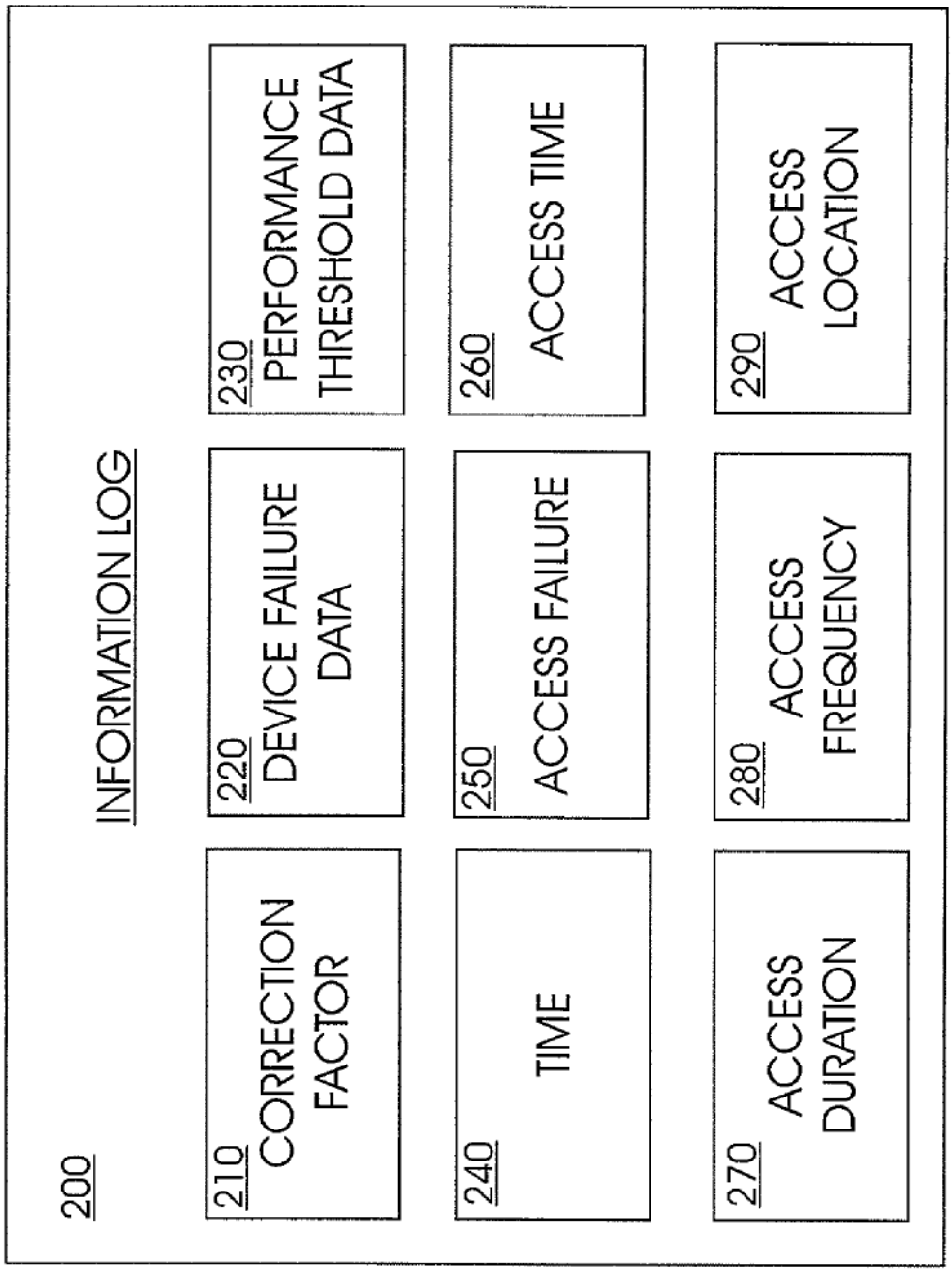

200
INFORMATION LOG
210 CORRECTION FACTOR
220 DEVICE FAILURE DATA
230 PERFORMANCE THRESHOLD DATA
240 TIME
250 ACCESS FAILURE
260 ACCESS TIME
270 ACCESS DURATION
280 ACCESS FREQUENCY
290 ACCESS LOCATION

FIG. 2

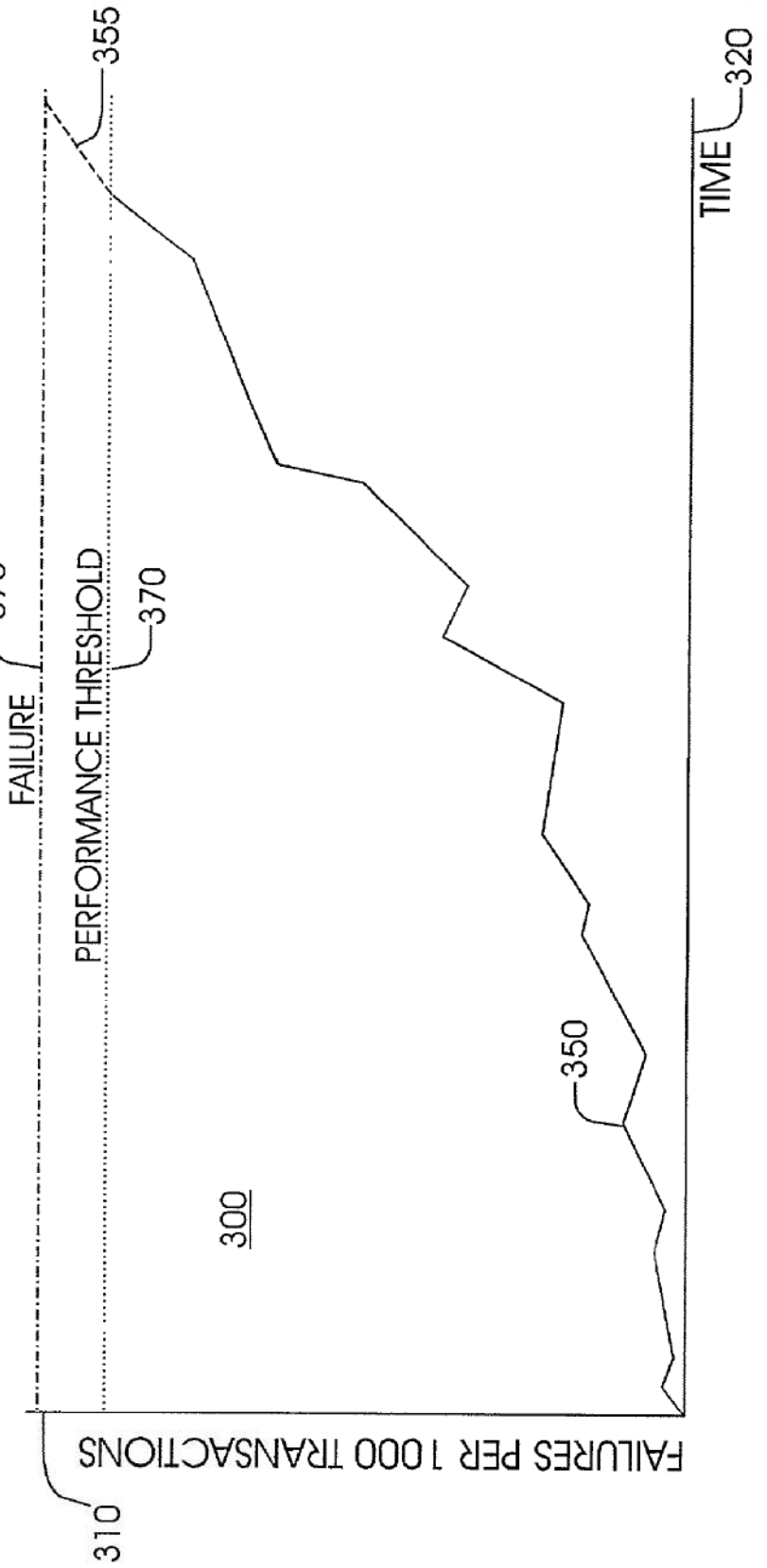

355
320
TIME
375
FAILURE
PERFORMANCE THRESHOLD
370
300
350
310
FAILURES PER 1 000 TRANSACTIONS

FIG. 3

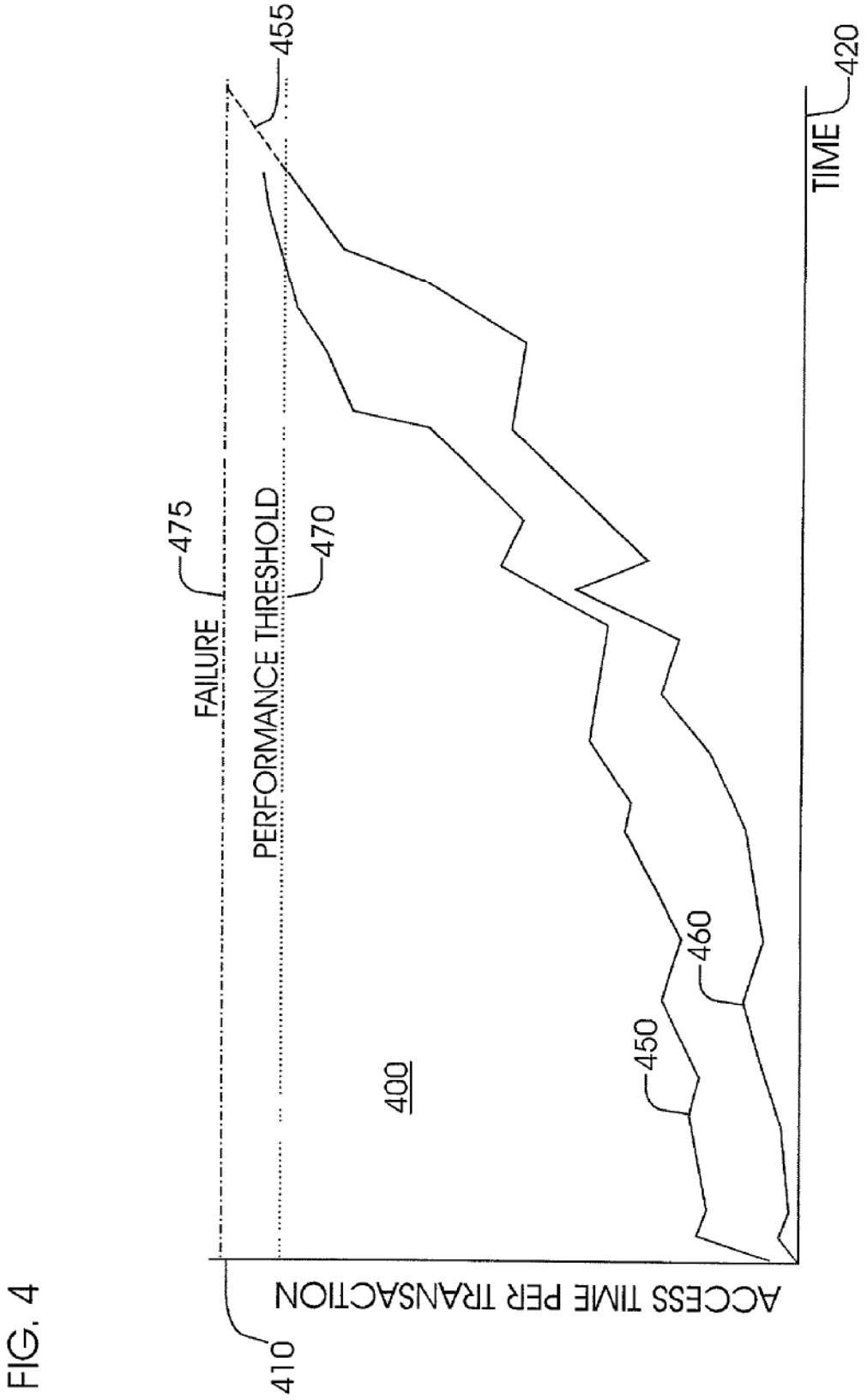

455
420
TIME
475
FAILURE
PERFORMANCE THRESHOLD
470
400
460
450
ACCESS TIME PER TRANSACTION
410

FIG. 4

FIG. 5

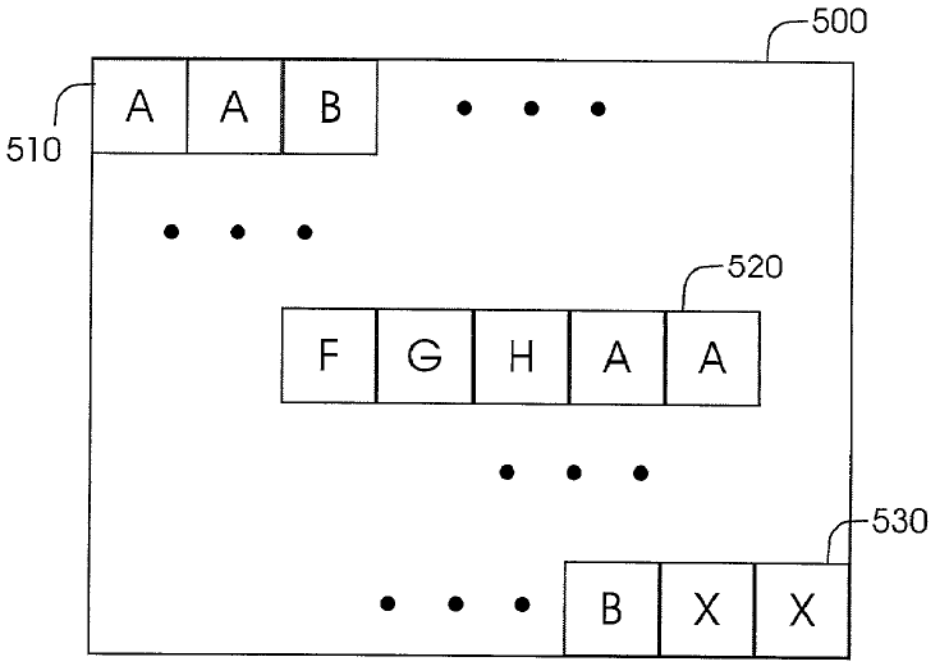


FIG. 6

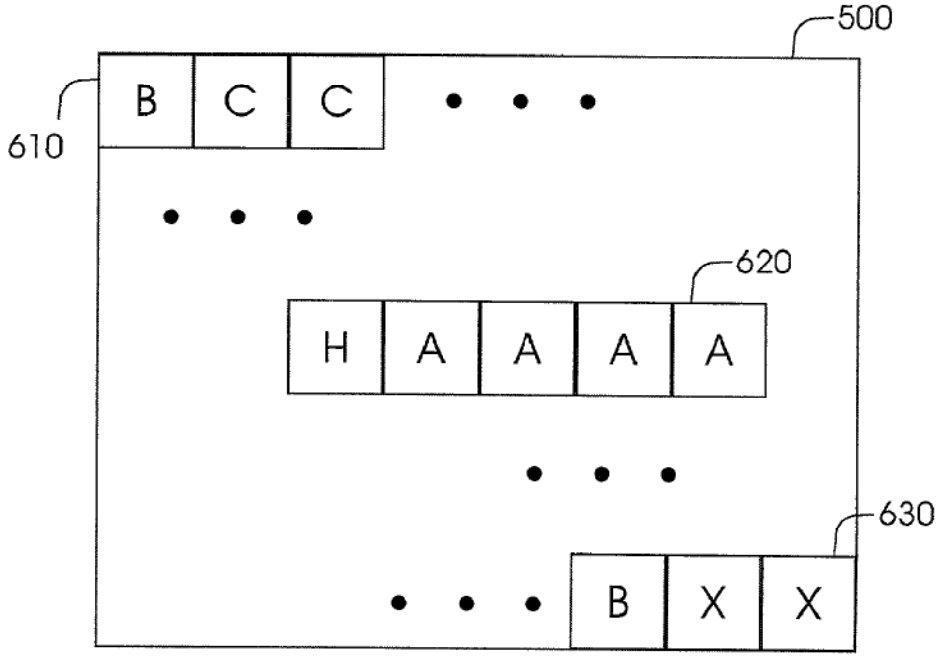


FIG. 7

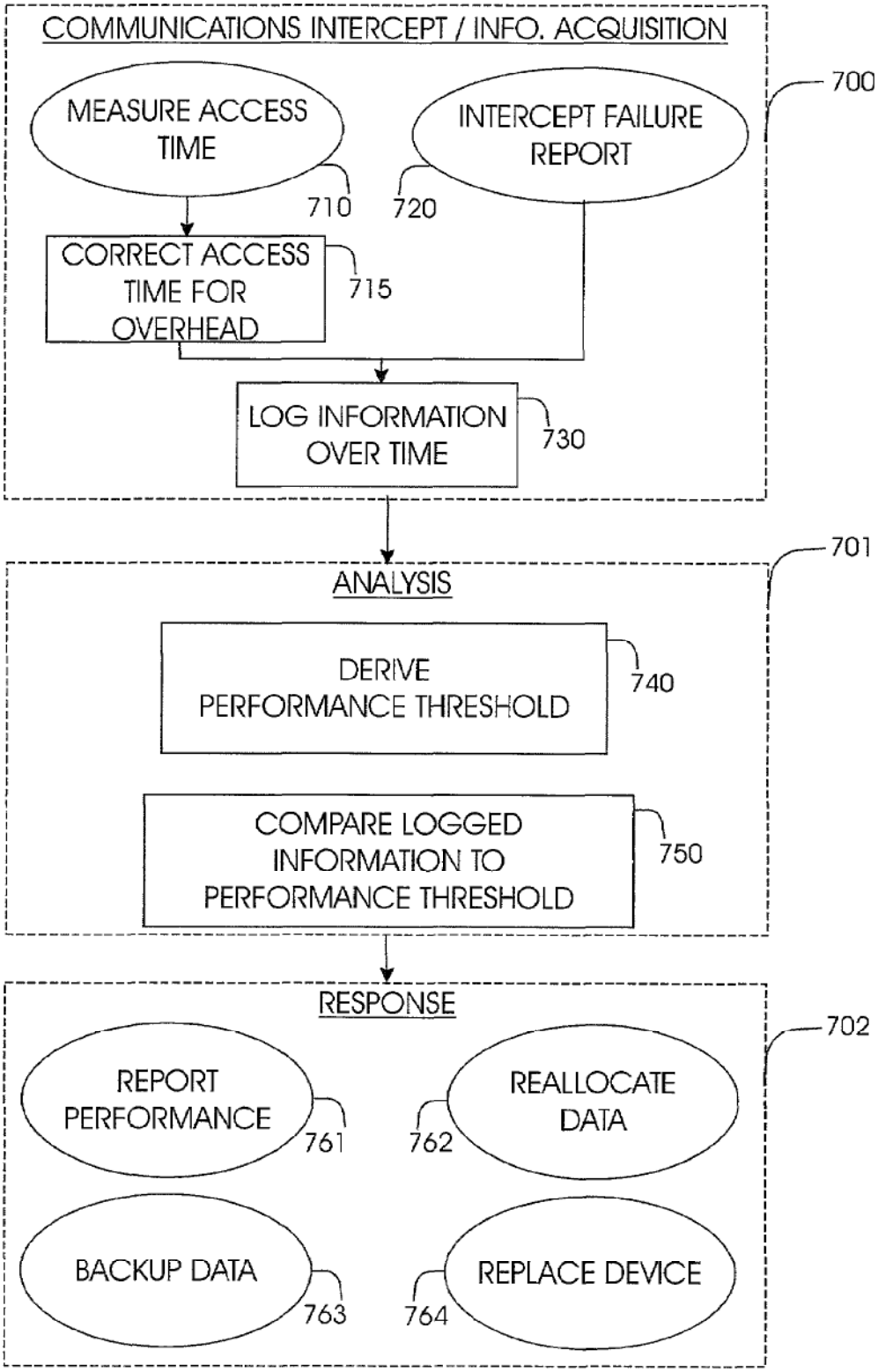
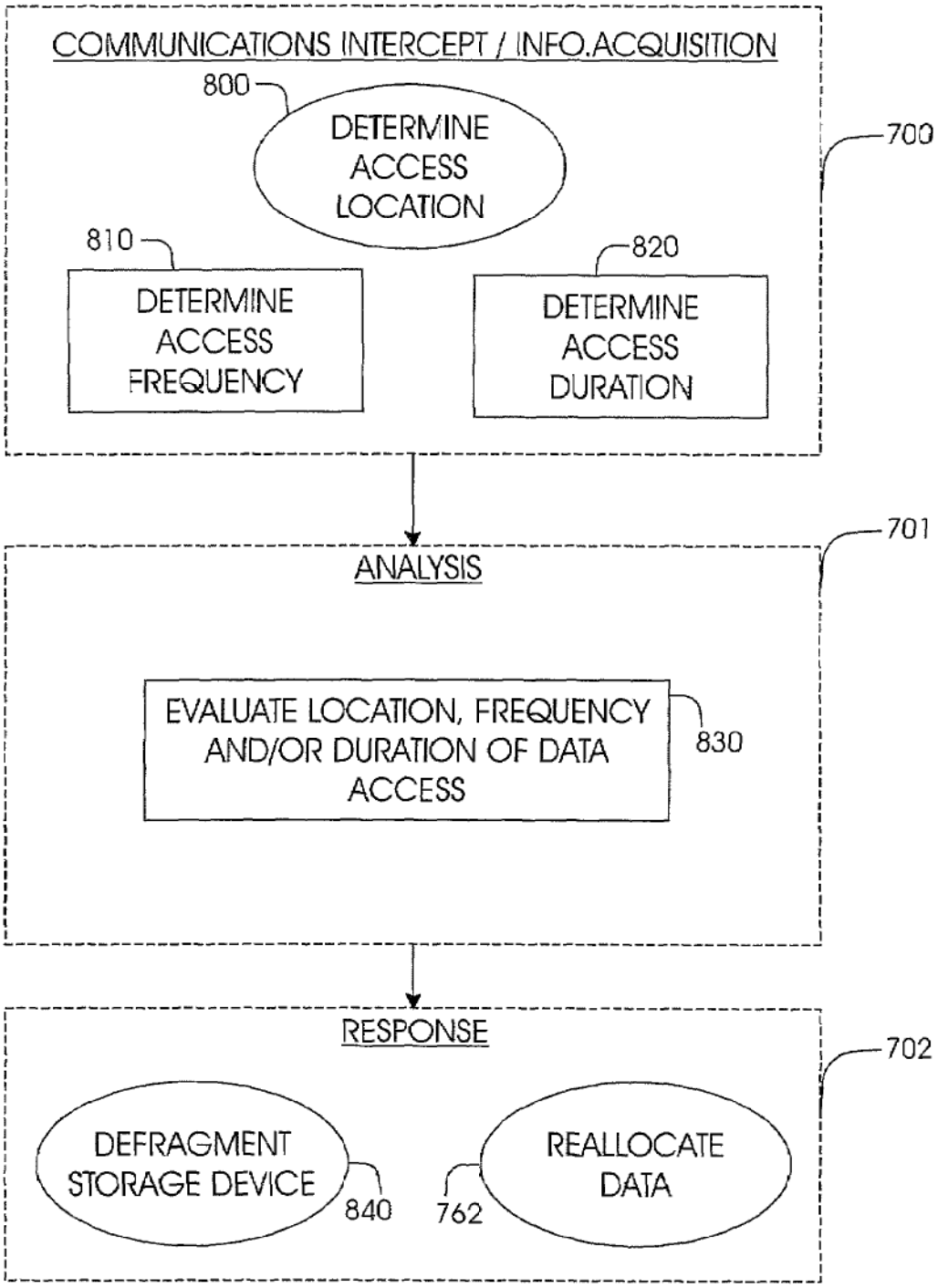

COMMUNICATIONS INTERCEPT / INFO. ACQUISITION
MEASURE ACCESS TIME
710
INTERCEPT FAILURE REPORT
720
700
CORRECT ACCESS TIME FOR OVERHEAD
715
LOG INFORMATION OVER TIME
730
701
ANALYSIS
DERIVE PERFORMANCE THRESHOLD
740
COMPARE LOGGED INFORMATION TO PERFORMANCE THRESHOLD
750
RESPONSE
702
REPORT PERFORMANCE
761
REALLOCATE DATA
762
BACKUP DATA
763
REPLACE DEVICE
764

FIG. 8


COMMUNICATIONS INTERCEPT / INFO.ACQUISITION
800
DETERMINE ACCESS LOCATION
700
810
DETERMINE ACCESS FREQUENCY
820
DETERMINE ACCESS DURATION
701
ANALYSIS
EVALUATE LOCATION, FREQUENCY AND/OR DURATION OF DATA ACCESS
830
RESPONSE
702
DEFRAGMENT STORAGE DEVICE
840
762
REALLOCATE DATA

STORAGE DEVICE PERFORMANCE MONITOR

FIELD OF THE INVENTION

The invention generally pertains to performance monitors for use with computer systems, and more specifically, to a method and apparatus for monitoring the performance of a storage device.

BACKGROUND OF THE INVENTION

Computer systems are typically provided with access to one or more storage devices. The storage devices may be attached directly to the computer system itself. For example, a personal computer (PC) such as a laptop or desktop computer may include one or more attached storage devices such as a hard disk drive, a compact disk (CD), magnetic storage, etc. Alternately, or in addition to, the computer system may have access to remote storage devices, such as over a network. For example, a workstation may have remote access to a network storage device such as a fibre channel Storage Area Network (SAN), a Network Attached Storage (NAS) device, etc. Or for example, a workstation may have remote access to a shared hard disk drive or other storage device attached to a server or other workstation that is available over a network.

The performance of a storage device, whether attached or remote, depends on a number of factors, such as the extent of use, type of use, etc. Over time and with use, the performance of the storage device may decline or even fail altogether. Declining performance and/or complete failure of a storage device may result in lost data, time and money, not to mention frustration for anyone needing access to the data stored thereon. In addition, when a storage device fails altogether, the data on the failed device may be irretrievable.

One solution for optimizing the performance of a storage device is to defragment it. During a defragmenting operation, similar files and file segments may be grouped together on the storage device so that these files and file segments may be more readily accessed. However, defragmenting operations often take time to perform, and may be interrupted if the storage device is accessed during the defragmenting operation. In addition, defragmenting operations typically group all similar or like files with one another, without first assessing the need to do. As such, a time consuming defragmenting operation may not improve the performance of the storage device.

When a storage device fails altogether, data recovery may include an attempt to recover the data from the failed storage device itself. In some situations, the data may be recovered. However, such recovery may take considerable time before the data can be retrieved from the failed storage device and rewritten to another storage device. The data remains inaccessible during the recovery operation. In addition, this solution may be costly depending on the type of storage media and the extent of the failure. In some circumstances, the data may not even be recoverable.

Another, more common solution for data recovery, is to back-up data from one storage device to another storage device prior to a device failure. However, the user does not know when a storage device will fail, and hence the user does not know when to perform the back-up operation. Device failures often start out as what are called "recoverable" failures. That is, when an attempt to access data on the storage device fails, the storage device itself may retry or make another attempt to access the data thereon. Alternately, the storage device may report the failed attempt to the operating system, which may retry or make another call to the storage device to access the data thereon. In any event, when a retry is successful, nothing is reported to the user. As such, any potential problems with the storage device are "masked" to the user, and the user does not know of a potential failure of the storage device until it actually occurs.

Typically, the user must back-up data on the storage device to another storage device on a regular basis. However, even when data is backed-up on a regular basis, the user still does not know when the storage device will fail. As such, some data may still be lost when the storage device fails. That is, the data added or changed after the last back-up may not be recoverable when the storage device fails. Although more frequent back-ups may reduce the amount of lost data when the storage device fails, back-up operations take time to perform, and may slow other functions of the computer system while being performed. In addition, there may still be some data that is changed and/or added after the last back-up, and thus that data may be lost when the storage device fails.

SUMMARY OF THE INVENTION

The inventors have devised an apparatus for monitoring performance of a storage device. The apparatus is preferably embodied in computer readable program code that is stored on computer readable storage media. The apparatus may comprise program code for intercepting communications between the storage device and a computer system; program code for analyzing the intercepted communications, wherein the intercepted communications are compared to a predicted failure of the storage device; and program code for responding to a decline in the performance of the storage device prior to the predicted failure thereof. The program code for intercepting the communications may comprise program code for intercepting an error reported by the storage device, program code for measuring access time for the storage device, and/or program code for determining system overhead and program code for correcting the access time for the system overhead. The program code for responding may comprise, for example, program code for backing-up data from the storage device, and/or reallocating data on another sector of the storage device. As another example, the program code for responding to the decline in the performance of the storage device may comprise defragmenting at least a portion of the storage device.

A method for monitoring performance of a storage device is also disclosed. The method may comprise intercepting communications between the storage device and a computer system; analyzing the intercepted communications relative to a threshold value for the performance of the storage device; and responding to a decline in the performance of the storage device based on the analyzed communications. Intercepting the communications may comprise measuring access time for the storage device, correcting the measured access time for system overhead, and/or intercepting a failure report from the storage device. Responding to the declining performance of the storage device may comprise, for example, automatically backing-up data stored on the storage device, reallocating data to other sectors on the storage device, replacing the storage device, reporting the performance (e.g., to an administrator), defragmenting the storage device, etc.

Accordingly, the storage device may be defragmented and/or the files and file segments are reallocated thereon after a determination that such action is necessary to

improve the performance of the storage device. In addition, the apparatus and method may determine how best to defragment and/or reallocate the storage device to optimize the performance thereof. Furthermore, the apparatus and method may monitor the performance of the storage device so that a response may be initiated prior to an actual failure of the storage device. For example, the data stored thereon may be moved prior to a failure, eliminating the need for expensive and time-consuming recovery operations.

BRIEF DESCRIPTION OF THE DRAWINGS

Illustrative and presently preferred embodiments of the invention are illustrated in the drawings, in which:

FIG. **1** is a high level diagram of a computer system and a storage device, illustrating an embodiment of the apparatus for monitoring the performance of the storage device;

FIG. **2** illustrates an information log for monitoring the performance of the storage device;

FIG. **3** shows exemplary information intercepted for analysis according to an embodiment of the apparatus;

FIG. **4** shows exemplary information intercepted for analysis according to another embodiment of the apparatus;

FIG. **5** illustrates data stored on various sectors of the storage device;

FIG. **6** illustrates data segments stored on various sectors of the storage device after the data has been reallocated or defragmented;

FIG. **7** illustrates embodiments of a method for monitoring the performance of the storage device; and

FIG. **8** illustrates further embodiments of a method for monitoring the performance of the storage device.

DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. **1** is a high level diagram illustrating an embodiment of the apparatus for monitoring the performance of a storage device **150**. For purposes of illustration, the computer system **100** is shown comprising at least an operating system **110** and may further comprise applications **120** executed by the computer system **100**. Although the storage device **150** is shown separately from the computer system **100**, these components may be housed in a single unit. Also shown in FIG. **1** is a filter driver **130** for intercepting **140** communications (e.g., write commands **160**, read commands **165**) between the computer system **100** (e.g., the operating system **110**, an application **120**) and the storage device **150**. In addition, the filter driver **130** may also intercept **145** errors or failures **170** reported by the storage device **150** to the computer system **100**. The information from the intercepted communications between the computer system **100** and the storage device **150**, and other information related thereto, as explained below, may be stored in an information log **200** for analysis thereof, according to the teachings of the invention.

The apparatus is preferably embodied in firmware and/or software (i.e., computer readable program code), stored in computer readable storage media and executed, for example, by the computer system **100**. The computer readable program code may comprise: program code for intercepting communications between the computer system **100** and the storage device **150**; program code for analyzing the intercepted communications, wherein the intercepted communications are compared to a predicted failure of the storage device **150**; and program code for responding to a decline in the performance of the storage device **150** prior to the predicted failure thereof.

The program code for intercepting the communications may comprise, or operate in conjunction with, the filter driver **130**. Communications that may be intercepted between the computer system **100** and the storage device **150** typically include "open", "close", "read", "write", "file creation", "error", etc. Mechanisms that allow applications to intercept such calls may be provided for use with operating systems, such as, Unix and Microsoft WINDOWS® operating systems. For example, see co-owned U.S. patent application Ser. No. 09/589,144 for "Efficient, Real-Time Management of Storage Resources", filed Jun. 7, 2000, of Duggan, et al., hereby incorporated by reference for all that it discloses.

It is understood that the computer system **100** may comprise any conventional computer, such as a desktop personal computer (PC), laptop PC, network workstation, network server, etc. In addition, the computer system **100** may comprise hardware and software that is not shown in FIG. **1**. It is also understood that the firmware and/or software may comprise more than one routine and/or subroutine, and may be embodied in separate components. In addition, the program code may be a stand-alone application, or may be a plug-in module for an existing application and/or operating system. Alternatively, the program code may be integrated into an application or operating system. In yet another embodiment, the program code can reside at one or more network devices (not shown), such as an administrator terminal, a server, etc.

For purposes of illustration, FIG. **1** shows a single storage device **150**. It is understood however, that the apparatus and method may be used with one or more storage devices **150**, or that a single storage device **150** may comprise one or more storage media (e.g., an array of disks). It is further understood that although the storage device **150** is preferably a hard disk drive, it can be any suitable fixed or removable computer readable storage medium. For example, the storage device **150** may be, but is not limited to, a hard disk drive, zip disk, compact disc (CD), magnetic tape, etc. In addition, the storage device **150** may be any suitable network storage device including, but not limited to, storage attached to a personal computer (PC) or server, etc. It is further understood that the nature of the storage device **150** is immaterial to the teachings of the invention. That is, the storage devices **110**–**113** may be shared, password protected, etc.

FIG. **2** illustrates an information log **200** for monitoring the performance of the storage device **150**. The information log **200** may be one or more suitable databases, data arrays, ASCII files, etc., for storing information (e.g., **240**–**290**) that is intercepted from the communications between the computer system **100** and the storage device **150**. Additional fields may also be provided, for example, for a correction factor **210** (e.g., for correcting the access time for overhead), device failure data **220** (e.g., for determining a threshold for responding to a decline in performance), and performance threshold data **230** (e.g., for responding to a decline in the performance of the storage device **150**, prior to failure thereof). Preferably, the information log **200** may comprise intercepted information such as time **240** (e.g., a specific time or a period of time), reported access failures **250** (e.g., at time **240**), and determined access times **260** (e.g., also at time **240**), for monitoring the performance of the storage device **150** and responding to a decline in performance thereof prior to a failure of the storage device **150**, as

discussed below. The information log **200** preferably also may comprise intercepted information such as access duration **270**, access frequency **280**, and access location **290**, for monitoring the performance of the storage device **150** and responding to a decline in performance thereof by defragmenting all or a portion of the storage device **150**, as discussed below.

According to one embodiment of the apparatus and method, the storage device **150** is monitored for "hidden" or "masked" signs of declining performance. That is, when a recoverable failure occurs during an attempt to access the storage device **150**, the storage device **150** may report an initial failure to the calling system or the operating system, which then retries the access command (e.g., an input/output (I/O) command) one or more times. When one of the retries is successful, nothing is reported to the user, thus masking a potential or pending problem with the storage device **150**.

According to the teachings of the invention, such a recoverable failure may be identified based on an analysis of the failures reported from the storage device to the computer system **100**. That is, the filter driver **130** may intercept reported errors or failures **170**. In addition, the filter driver **130** and/or other suitable program code may also intercept other communications, such as, the location of attempted access on the storage device **150**, amount of data, type of access, duration of access, etc. The information from the intercepted communications may be written to a storage database (e.g., information log **200**). Suitable program code may also be provided for analyzing the intercepted communications. For example, the program code for analyzing the intercepted communications may find an increasing number of failures for the storage device **150** and/or a particular area thereof. When the number of failures exceeds a threshold, program code for responding to the decline in performance of the storage device **150** may warn the user of a potential or pending problem with the storage device **150**.

FIG. **3** shows exemplary information intercepted from the communications between the computer system **100** and the storage device **150**, for analysis according to the embodiment of the apparatus and method where a failure is reported from the storage device **150**. This embodiment is further illustrated by the data presented in Table 1.

TABLE 1

Time	Failures Per 1000 Transactions
T1	0
T2	3
T3	1
. . .	. . .
T100	77

In this example, the number of failures per thousand transactions is intercepted and logged over time. For purposes of illustration, the data is shown by plot **300** in FIG. **3**, wherein the number of failures per thousand transactions is plotted along the y-axis **310** as a function of time, shown along the x-axis **320**. The logged information **350** indicates that during early operations of the storage device **150**, there are very few failures reported by the storage device **150** to the computer system **100**. For example, at times T**1**, T**2**, and T**3** (see Table 1, above), there are very few failures reported. However, with use and over time, the logged communications **350** indicates that the storage device **150** begins to report more access failures. For example, at time T**100** (see Table 1, above), there are seventy-seven failures reported for every one-thousand transactions that occur.

According to the invention, the intercepted communications may be compared to a known or predicted failure **375** of the storage device **150**. That is, based on past performance of comparable storage devices, it may be known that the storage device **150** may fail entirely when the storage device **150** experiences a number of recoverable failures (e.g., approximately 100 failures for every one-thousand transactions). Alternately, or in addition to, the failure **375** may be derived or predicted to fail entirely when the storage device **150** experiences a number of recoverable failures (e.g., approximately 100 failures for every one-thousand transactions), based on statistical analysis of the logged information (e.g., curve fit **355**).

A performance threshold **370** may be determined based on the known or predicted failure **375**. Thus, when the analysis of the intercepted communications indicate that the performance of the storage device **150** is approximately at the performance threshold **370**, a response may be initiated to prevent loss of the data on the storage device **150**. For example, a performance threshold **375** may be "seventy-five failures per thousand transactions", based on the predicted failure **375** of "one-hundred failures per thousand transactions". Thus, for example, when the intercepted communications indicate that the performance of the storage device **150** is approximately seventy-five failures per thousand transactions (e.g., seventy-seven reported failures at time T**100** in Table 1, above), a response may comprise alerting the user that the storage device **150** is about to fail so that the user may replace it, automatically backing-up or reallocating data on the storage device **150** to an alternate storage device, etc. As such, a response may be initiated prior to failure of the storage device **150** to prevent, or reduce the risk of, losing the data on the storage device **150** due to a failure thereof.

It is understood that the data shown in FIG. **3** and the data shown in Table 1 is merely illustrative of information that may be intercepted from communications between the computer system **100** and the storage device **150**, and analyzed according to the teachings of the invention. The examples given above with respect to FIG. **3** and Table 1 are not intended to limit the scope of the intercepted communications to the values of this information or the type of information that may be intercepted and analyzed according to the invention. In addition, these examples are not intended to limit the teachings of the invention in any other manner. Furthermore, the information of FIG. **3** and Table 1 is not to be construed as actual data, and was not measured as such.

According to another embodiment, the storage device **150** is monitored for "hidden" or "masked" signs of declining performance. That is, when a recoverable failure occurs during an attempt to access the storage device **150**, the storage device **150** may retry the access command (e.g., an input/output (I/O) command) itself. When the retry is successful, nothing is reported back up the calling chain (i.e., to the computer system **100**), thus masking a potential or pending problem with the storage device **150**. Because the recoverable error or failure is not reported by the storage device **150**, however, the error or failure is not directly viewable by the filter driver **130**.

According to the teachings of the invention, such a recoverable failure may be inferred by analyzing a history of access times to various parts of the storage device **150**. That is, when the storage device **150** retries the access command (e.g., up to 10 times), a timer may be incremented until the access is successful. In addition, the filter driver **130** and/or

other suitable program code may also intercept other communications, such as, the location of attempted access on the storage device, amount of data, type of access, duration of access, etc. The intercepted information may be written to a storage database (e.g., information log **200**). Suitable program code may also be provided for analyzing the intercepted communications. For example, the program code for analyzing the intercepted communications may find lengthening execution or access times for the storage device **150** and/or a portion thereof. When the access time exceeds a threshold, program code for responding to the decline in performance of the storage device **150** may warn the user of a potential or pending problem with the storage device **150**.

FIG. **4** shows exemplary information intercepted between the computer system **100** and the storage device **150**, for analysis according to the embodiment where an error or failure is not reported from the storage device **150**. This embodiment is further illustrated by the data presented in Table 2.

TABLE 2

Time	Measured Access Time (ms)	Corrected Access Time (ms)
T1	0.3	0.1
T2	0.3	0.2
T3	0.4	0.3
. . .	. . .	. . .
T100	2.8	2.6

In this example, the access time for the storage device **150** is intercepted and logged over time. For purposes of illustration, the data is shown by plot **400** in FIG. **4**, wherein the access time is plotted along the y-axis **410** as a function of time, shown along the x-axis **420**. The logged data may comprise measured access time **450** and/or corrected access time **460**. The exemplary logged data indicates that during early operations of the storage device **150**, the access time for the storage device **150** is relatively fast. For example, at times T**1**, T**2**, and T**3** (see Table 2, above), the access time is less than 0.5 ms. However, with use and over time, the data **450** indicates that the storage device **150** begins to slow. For example, at time T**100** (see Table 2, above), the average access time exceeds 2.5 ms, even once it has been corrected.

The access time for the storage device **150** may be influenced by a number of external factors, in addition to internal retries. For example, more than one call may be made simultaneously to the storage device **150**, the load on the computer system **100** may impact the communications with the storage device **150**, etc. Or for example, the overhead may be generated by, but not limited to, the CPU, the I/O overhead lock, IRQ lock, device I/O queue management, network load (e.g., for network accessed drives). These, and/or other external factors may influence the access time of the storage device **150** at various times, and indeed, need not influence the access time at all during other times.

Therefore, the apparatus preferably comprises program code for correcting the measured access time for these external factors. For example, the filter driver **130** may also monitor other calls to the storage device **150**, and based on the number of simultaneous calls thereto, apply a correction factor to determine the access time of the storage device **150**. Or for example, the operating system **110** may report the load on the computer system **100**, and based on the load, a correction factor may be applied to determine the access time of the storage device **150**. Or for example, the access time may be measured as a result of the filter driver itself sifting at a higher level and priority in processing. It may then determine when the storage device **150** is being accessed for a file versus when it is busy with an overhead function. These are merely exemplary of solutions for correcting the measured access time therefor, and other solutions thereto are also contemplated as being within the scope of the invention.

According to the invention, the intercepted communications may be compared to a known or predicted failure **475** of the storage device **150**. That is, based on past performance of comparable storage devices, it may be known that the storage device **150** may fail entirely when the access time for the storage device **150** slows to a known or expected value (e.g., 3.0 ms). Alternately, or in addition to, the failure **475** may be derived or predicted to fail entirely when the storage device **150** slows to a known or expected value (e.g., 3.0 ms), based on statistical analysis of the intercepted communications (e.g., curve fit **455**). Thus, a performance threshold **470** may be determined based on the known or predicted failure **475**. Accordingly, when the intercepted communications indicate that the performance of the storage device **150** is approximately the performance threshold **370**, a response may be initiated to prevent loss of the data on the storage device **150**. For example, a performance threshold **475** may be an access time of 2.5 ms, based on the predicted failure of an access time of 3.0 ms. Thus, for example, when the intercepted communications indicate that the performance of the storage device **150** has an access time of approximately 2.5 ms (e.g., 2.6 ms at time T**100** in Table 2, above), a response may be initiated. A response may comprise alerting the user that the storage device **150** is about to fail so that the user may replace the device, automatically backing-up or reallocating data on the storage device **150** to an alternate storage device, etc. As such, a response may be initiated prior to failure of the storage device to prevent, or reduce the risk of, losing the data on the storage device **150** due to a failure thereof.

It is understood that the data shown in FIG. **4** and the intercepted information shown in Table 2 is merely illustrative of communications that may be intercepted and analyzed. The examples given above with respect to FIG. **4** and Table 2 are not intended to limit the communications to the values thereof, or to the type of communications that may be intercepted and analyzed. In addition, these examples are not intended to limit the teachings of the invention in any other manner. Furthermore, the data of FIG. **4** and Table 2 is not to be construed as actual information, and was not measured as such.

Another embodiment may comprise defragmenting at least a part of the storage device **150** in response to the declining performance thereof. A storage device may become fragmented with use. However, when the fragmentation occurs only with respect to some of the data thereon, it may be inefficient to defragment the entire storage device **150**. In addition, when the fragmentation occurs only with respect to data that is rarely accessed, it may be inefficient to defragment the storage device **150** at all. Therefore, the invention contemplates efficiently defragmenting the storage device **150**. For example, directories may be reallocated to sectors that are closer to the files each represents. Or for example, directories may be grouped on the storage device **150** in the middle of the device to increase the access speed thereof. Or as another example, the most frequently used files may be reallocated to sectors in or near the center of the storage device **150** to increase the access speed thereof. These, and other examples of efficiently defragmenting the storage device **150** are contemplated under the teachings of

the invention and are discussed in more detail below with respect to FIG. **5** and FIG. **6**.

FIG. **5** illustrates a portion **500** of the storage device **150** having data (e.g., A, B, F, G, etc.) stored on various sectors thereof (e.g., blocks of sectors **510**, **520**, **530**). The filter driver **130** and/or other suitable program code intercepts calls (e.g., **160**, **165**) to and from the storage device **150**, and thus intercepts communications, such as, the file or file segment identity, the frequency with which each file or file segment is accessed, the location of each attempted accesses, the duration that each file or file segment is accessed, etc. According to the invention, this information may be stored in a suitable database (e.g., information log **200**). Exemplary information that may be intercepted for the files and/or file segments shown in FIG. **5** is given in Table 3, below.

TABLE 3

File	File Location-Sector(s)	Access Duration-Avg Time (sec)	Access Frequency-(No. Per Hour)
A	1, 2, 450, 451	188	550
B	3, 1001	15	2
. . .	. . .	. . .	. . .
F	447	78	15
G	448	158	55
H	449	255	351
. . .	. . .	. . .	. . .
X	1002, 1003	37	7

The exemplary communications in Table 3 indicates that the data "A" (e.g., a file, file segment, directory, etc.) resides at four different sectors on the storage device **150**. In addition, the average duration of each access of the data "A" is relatively significant (i.e., 188 seconds), as is the average access frequency thereof (i.e., 550 times per hour). The data "B" resides at two different sectors on the storage device **150**, which are identified by the intercepted communications. In addition, the average duration of each access of the data "B" is relatively insignificant (i.e., 15 seconds), as is the average frequency with which it is accessed (i.e., twice per hour). Therefore, an analysis of this information may indicate that the data "A" may be reallocated, while the data "B" need not necessarily be reallocated at this time, to efficiently defragment the storage device **150**.

The intercepted communications and analysis thereof may be used by the invention to map the storage device **150**, or portions thereof (e.g., portion **500**), and to reallocate data thereon for optimal access thereto for the average use. FIG. **6** illustrates data (e.g., A, B, F, G, etc.) stored on various sectors (e.g., blocks of sectors **610**, **620**, **630**) of the portion **500** of the storage device **150** after it has been at least partially defragmented and/or the data has been reallocated thereon. For example, based on the above analysis of the data in Table 3, the data "A" may be reallocated to adjacent or nearby sectors on the storage device **150**. In addition, the data "A" may also be reallocated to reside at or near the center of the storage device **150** (e.g., to the block of sectors **620**). As such, the access speed of data "A" may be increased and the storage device **150** may be efficiently defragmented.

It is understood that the examples given above with respect to FIG. **5**, FIG. **6**, and Table 3 are provided to illustrate various embodiments of the invention. In addition, it is understood that the representations of sectors of the storage device **150** shown in FIG. **5** and FIG. **6**, and the information in Table 3 are not to be construed as actual representations and/or data, and was not measured or otherwise determined as such.

It is also understood that the examples given above with respect to FIG. **5**, FIG. **6**, and Table 3 are not intended to limit the scope of the invention thereto. As another example, defragmentation, or partial defragmentation of the storage device **150** may be based on usage patterns. The intercepted communications and analysis thereof may indicate that directly accessed data may be associated with other data on various sectors. For example, when a particular file is opened (e.g., by the user), other associated files are also opened (e.g., by the system). As such, the directly accessed data and the associated data may be reallocated to adjacent or nearby sectors on the storage device **150**. As yet another example, the invention may also be used to defragment more than one storage device. Likewise, the data may be reallocated and/or defragmented across more than one such storage device, or among more than one media within a single storage device (e.g., between partitioned drives). Other embodiments are also contemplated as being within the scope of the invention.

FIG. **7** illustrates embodiments of a method for monitoring the performance of the storage device **150**. Generally, the method includes intercepting communications between the computer system **100** and the storage device **150** to acquire information (**700**), analysis of the intercepted information (**701**), and response to the analysis of the intercepted information (**702**). More specifically, acquisition **700** may comprise measuring the access time in step **710**, and optionally correcting the access time for overhead in step **715**. The access time may also be logged overtime, in step **730** (e.g., in the information log **200**). In another embodiment, acquisition **700** may comprise intercepting an error or failure reported by the storage device **150**, and optionally logging the error or failure over time, in step **730**. Analysis **701** may comprise deriving a performance threshold (e.g., **370**, **470**) in step **740**, and/or comparing the logged information to a performance threshold in step **750**. The response **702** may comprise reporting the performance of the storage device **150** in step **761**, and/or automatic back-up of the data on the storage device **150** (e.g., prior to failure thereof) in step **763**. Alternately, or in addition to, response **702** may comprise reallocating all or some of the data to another portion of the storage device **150** (e.g., to another sector), to another storage device **150**, both in step **702**, and/or replacing the storage device altogether, in step **764**.

FIG. **8** illustrates other embodiments of a method for monitoring the performance of the storage device **150**. Again, the method generally includes intercepting communications between the computer system **100** and the storage device **150** to acquire information (**700**), analysis of the intercepted information (**701**), and response to the analysis of the intercepted information (**702**). However, in the embodiments shown in FIG. **8**, acquisition **700** may comprise determining the access location(s) for data on the storage device **150** in step **800**. In step **810**, the access frequency of the data may be determined. In addition, or instead of, acquisition **700** may comprise determining the access duration of the data in step **820**. Analysis **701** may comprise evaluating the location, frequency, and/or duration of access of the data in step **830**. Response **702** may comprise defragmenting the storage device, in step **840**. Alternatively, or in addition thereto, data may be reallocated in step **762**.

It is understood that the steps shown and described in FIG. **7** and in FIG. **8** are merely illustrative of various exemplary embodiments of the invention and the scope of the invention is not to be limited thereto. The invention may comprise other embodiments having additional or fewer steps than those shown and described above.

The invention has been described above and various embodiments thereof have been illustrated for exemplary purposes. It is understood, however, that other embodiments are also contemplated as being within the scope of the invention. For example, another embodiment may comprise a graphical user interface (GUI) with graphical and/or tabular representations of the intercepted communications and/or analysis thereof. Or for example, in another embodiment, the invention may output a detailed view of the storage device and the performance thereof under various conditions (e.g., under various loads, with respect to certain applications and/or users, etc.). In yet another exemplary embodiment, the invention may output the results of a competitive analysis that may be used to evaluate the performance of existing storage devices **150**, storage devices under development, storage devices under consideration for purchase, etc. Yet other embodiments will readily occur to those skilled in the art in view of the teachings of the invention.

While illustrative and presently preferred embodiments of the invention have been described in detail herein, it is to be understood that the inventive concepts may be otherwise variously embodied and employed, and that the appended claims are intended to be construed to include such variations, except as limited by the prior art.

What is claimed is:

1. A method for monitoring performance of a storage device, comprising:

intercepting communications between a computer system and said storage device;

analyzing said intercepted communications relative to a threshold value for the performance of said storage device; and

responding to a decline in the performance of said storage device based on said analyzed intercepted communications by automatically reallocating at least some data on said storage device to enhance continued oneration of the storage device.

2. A method as in claim **1**, further comprising measuring access time for said storage device.

3. A method as in claim **2**, further comprising correcting said measured access time for system overhead.

4. A method as in claim **1**, wherein intercepting said communications comprises intercepting an error reported by said storage device.

5. A method as in claim **1**, further comprising determining an access location on said storage device and an access frequency for data stored thereon, based on said intercepted communications.

6. A method as in claim **1**, further comprising determining an access location on said storage device and an access duration for data stored thereon, based on said intercepted communications.

7. A method as in claim **1**, further comprising logging said communications over time.

8. A method as in claim **7**, wherein analyzing said communications comprises deriving said threshold value based on said logged communications.

9. A method as in claim **1**, wherein responding to said declining performance of said storage device further comprises automatically backing-up data stored on said storage device.

10. A method as in claim **1**, wherein reallocating at least some data on said storage device is based on usage patterns of said data.

11. A method as in claim **1**, wherein responding to said declining performance of said storage device further comprises defragmenting at least a portion of said storage device.

12. An apparatus for monitoring performance of a storage device, comprising:

computer readable storage media;

computer readable program code scored on said computer readable storage media, comprising;

a) program code for intercepting communications between a computer system and said storage device;

b) program code for analyzing said communications, wherein said communications are compared to a predicted decline in performance of said storage device; and

c) program code for responding to a decline in the performance of said storage device by backing up at least some data on said storage device prior to said predicted failure of said storage device to enhance continued operation of said storage device.

13. An apparatus as in claim **12**, wherein said program code for intercepting said communications comprises program code for intercepting an error reported by said storage device.

14. An apparatus as in claim **12**, further comprising program code for measuring access time for said storage device, and wherein said communications comprises at least said access time.

15. An apparatus as in claim **14**, further comprising:

a) program code for determining system overhead; and

b) program code for correcting said access time for said system overhead.

16. An apparatus as in claim **12**, wherein said program code for responding to said decline in the performance of said storage device further comprises defragmenting at least a portion of said storage device based on said communications.

17. An apparatus as in claim **12**, further comprising:

a communications log;

program code for logging said communications over time in said communications log; and

program code for determining the decline in performance of said storage device based at least in part on said logged communications.

18. An apparatus as in claim **12**, further comprising program code for deriving a threshold value for the performance of said storage device, wherein said program code for responding to said decline in the performance of said storage device responds when the performance of said storage device satisfies said threshold value thereof.

19. An apparatus as in claim **12**, wherein said program code for responding further comprises program code for reallocating data to another sector of said storage device.

20. An apparatus as in claim **19**, wherein said program code for reallocating reallocates at least same data on said storage device based on usage patterns of said a least some data.

21. An apparatus as in claim 12, further comprising a graphical user interface for reporting the performance of said storage device to a user.

22. An apparatus for monitoring performance of a storage device, comprising:

means for evaluating communications between a computer system and said storage device to determine the performance of said storage device based at least in part on intercepted communications with said storage device; and

means for enhancing the performance of said storage device by reallocating at least some data on the storage device.

23. An apparatus as in claim 22, further comprising means for intercepting communications with said storage device.

24. An apparatus as in claim 22, further comprising means for measuring time to access said storage device, wherein said measured access time is evaluated to determine the performance of said storage device.

25. A method for monitoring performance of a storage device, comprising:

intercepting communications between a computer system and said storage device;

analyzing said intercepted communications; and

reallocating at least some of said data on said storage device to enhance the performance of said storage device based on said analyzed communications.

26. A method as in claim 25, wherein intercepting said communications comprises determining access location, access frequency, and access duration for said data on said storage device.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 7,120,832 B2
APPLICATION NO. : 09/966953
DATED : October 10, 2006
INVENTOR(S) : Kevin Collins et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In column 8, line 1, delete "sifting" and insert -- sitting --, therefor.

In column 11, line 45, in Claim 1, delete "oneration" and insert -- operation --, therefor.

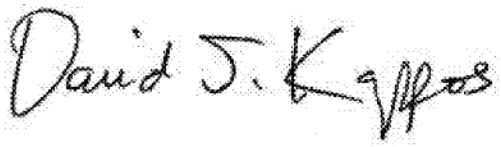
In column 12, line 16, in Claim 12, delete "scored" and insert -- stored --, therefor.

In column 12, line 17, in Claim 12, delete "comprising;" and insert -- comprising: --, therefor.

In column 12, line 65, in Claim 20, delete "same" and insert -- some --, therefor.

In column 12, line 66, in Claim 20, delete "a" and insert -- at --, therefor.

Signed and Sealed this
Tenth Day of July, 2012



David J. Kappos
Director of the United States Patent and Trademark Office

JS 44 (Rev. 03/24)

CIVIL COVER SHEET

The JS 44 civil cover sheet and the information contained herein neither replace nor supplement the filing and service of pleadings or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. *(SEE INSTRUCTIONS ON NEXT PAGE OF THIS FORM.)*

I. (a) PLAINTIFFS

Valtrus Innovations Ltd. and
Key Patent Innovations Ltd.

(b) County of Residence of First Listed Plaintiff Outside of U.S.
(EXCEPT IN U.S. PLAINTIFF CASES)

(c) Attorneys *(Firm Name, Address, and Telephone Number)*

Fabricant LLP, 411 Theodore Fremd Avenue, Suite 206S, Rye NY, 10580; (212) 257-5797

DEFENDANTS

FedEx Corporation and FedEx Office and Print Services, Inc.

County of Residence of First Listed Defendant
(IN U.S. PLAINTIFF CASES ONLY)

NOTE: IN LAND CONDEMNATION CASES, USE THE LOCATION OF THE TRACT OF LAND INVOLVED.

Attorneys *(If Known)*

II. BASIS OF JURISDICTION *(Place an "X" in One Box Only)*

- ☐ 1 U.S. Government Plaintiff
- ☒ 3 Federal Question *(U.S. Government Not a Party)*
- ☐ 2 U.S. Government Defendant
- ☐ 4 Diversity *(Indicate Citizenship of Parties in Item III)*

III. CITIZENSHIP OF PRINCIPAL PARTIES *(Place an "X" in One Box for Plaintiff and One Box for Defendant)*
(For Diversity Cases Only)

	PTF	DEF		PTF	DEF
Citizen of This State	☐ 1	☐ 1	Incorporated *or* Principal Place of Business In This State	☐ 4	☐ 4
Citizen of Another State	☐ 2	☐ 2	Incorporated *and* Principal Place of Business In Another State	☐ 5	☐ 5
Citizen or Subject of a Foreign Country	☐ 3	☐ 3	Foreign Nation	☐ 6	☐ 6

IV. NATURE OF SUIT *(Place an "X" in One Box Only)*

Click here for: Nature of Suit Code Descriptions.

CONTRACT
- ☐ 110 Insurance
- ☐ 120 Marine
- ☐ 130 Miller Act
- ☐ 140 Negotiable Instrument
- ☐ 150 Recovery of Overpayment & Enforcement of Judgment
- ☐ 151 Medicare Act
- ☐ 152 Recovery of Defaulted Student Loans (Excludes Veterans)
- ☐ 153 Recovery of Overpayment of Veteran's Benefits
- ☐ 160 Stockholders' Suits
- ☐ 190 Other Contract
- ☐ 195 Contract Product Liability
- ☐ 196 Franchise

TORTS

PERSONAL INJURY
- ☐ 310 Airplane
- ☐ 315 Airplane Product Liability
- ☐ 320 Assault, Libel & Slander
- ☐ 330 Federal Employers' Liability
- ☐ 340 Marine
- ☐ 345 Marine Product Liability
- ☐ 350 Motor Vehicle
- ☐ 355 Motor Vehicle Product Liability
- ☐ 360 Other Personal Injury
- ☐ 362 Personal Injury - Medical Malpractice

PERSONAL INJURY
- ☐ 365 Personal Injury - Product Liability
- ☐ 367 Health Care/ Pharmaceutical Personal Injury Product Liability
- ☐ 368 Asbestos Personal Injury Product Liability

PERSONAL PROPERTY
- ☐ 370 Other Fraud
- ☐ 371 Truth in Lending
- ☐ 380 Other Personal Property Damage
- ☐ 385 Property Damage Product Liability

FORFEITURE/PENALTY
- ☐ 625 Drug Related Seizure of Property 21 USC 881
- ☐ 690 Other

LABOR
- ☐ 710 Fair Labor Standards Act
- ☐ 720 Labor/Management Relations
- ☐ 740 Railway Labor Act
- ☐ 751 Family and Medical Leave Act
- ☐ 790 Other Labor Litigation
- ☐ 791 Employee Retirement Income Security Act

IMMIGRATION
- ☐ 462 Naturalization Application
- ☐ 465 Other Immigration Actions

BANKRUPTCY
- ☐ 422 Appeal 28 USC 158
- ☐ 423 Withdrawal 28 USC 157

INTELLECTUAL PROPERTY RIGHTS
- ☐ 820 Copyrights
- ☒ 830 Patent
- ☐ 835 Patent - Abbreviated New Drug Application
- ☐ 840 Trademark
- ☐ 880 Defend Trade Secrets Act of 2016

SOCIAL SECURITY
- ☐ 861 HIA (1395ff)
- ☐ 862 Black Lung (923)
- ☐ 863 DIWC/DIWW (405(g))
- ☐ 864 SSID Title XVI
- ☐ 865 RSI (405(g))

FEDERAL TAX SUITS
- ☐ 870 Taxes (U.S. Plaintiff or Defendant)
- ☐ 871 IRS—Third Party 26 USC 7609

OTHER STATUTES
- ☐ 375 False Claims Act
- ☐ 376 Qui Tam (31 USC 3729(a))
- ☐ 400 State Reapportionment
- ☐ 410 Antitrust
- ☐ 430 Banks and Banking
- ☐ 450 Commerce
- ☐ 460 Deportation
- ☐ 470 Racketeer Influenced and Corrupt Organizations
- ☐ 480 Consumer Credit (15 USC 1681 or 1692)
- ☐ 485 Telephone Consumer Protection Act
- ☐ 490 Cable/Sat TV
- ☐ 850 Securities/Commodities/ Exchange
- ☐ 890 Other Statutory Actions
- ☐ 891 Agricultural Acts
- ☐ 893 Environmental Matters
- ☐ 895 Freedom of Information Act
- ☐ 896 Arbitration
- ☐ 899 Administrative Procedure Act/Review or Appeal of Agency Decision
- ☐ 950 Constitutionality of State Statutes

REAL PROPERTY
- ☐ 210 Land Condemnation
- ☐ 220 Foreclosure
- ☐ 230 Rent Lease & Ejectment
- ☐ 240 Torts to Land
- ☐ 245 Tort Product Liability
- ☐ 290 All Other Real Property

CIVIL RIGHTS
- ☐ 440 Other Civil Rights
- ☐ 441 Voting
- ☐ 442 Employment
- ☐ 443 Housing/ Accommodations
- ☐ 445 Amer. w/Disabilities - Employment
- ☐ 446 Amer. w/Disabilities - Other
- ☐ 448 Education

PRISONER PETITIONS

Habeas Corpus:
- ☐ 463 Alien Detainee
- ☐ 510 Motions to Vacate Sentence
- ☐ 530 General
- ☐ 535 Death Penalty

Other:
- ☐ 540 Mandamus & Other
- ☐ 550 Civil Rights
- ☐ 555 Prison Condition
- ☐ 560 Civil Detainee - Conditions of Confinement

V. ORIGIN *(Place an "X" in One Box Only)*

☒ 1 Original Proceeding ☐ 2 Removed from State Court ☐ 3 Remanded from Appellate Court ☐ 4 Reinstated or Reopened ☐ 5 Transferred from Another District *(specify)* ☐ 6 Multidistrict Litigation - Transfer ☐ 8 Multidistrict Litigation - Direct File

VI. CAUSE OF ACTION

Cite the U.S. Civil Statute under which you are filing ***(Do not cite jurisdictional statutes unless diversity)***:
35 U.S.C. §§ 1, et seq.

Brief description of cause:
Patent Infringement

VII. REQUESTED IN COMPLAINT:

☐ CHECK IF THIS IS A **CLASS ACTION** UNDER RULE 23, F.R.Cv.P.

DEMAND $

CHECK YES only if demanded in complaint:
JURY DEMAND: ☒ Yes ☐ No

VIII. RELATED CASE(S) IF ANY *(See instructions):*

JUDGE Rodney Gilstrap DOCKET NUMBER 2:25-cv-80 and 2:25-cv-81

DATE 5/9/2025

SIGNATURE OF ATTORNEY OF RECORD /s/ Vincent J. Rubino III

FOR OFFICE USE ONLY

RECEIPT # AMOUNT APPLYING IFP JUDGE MAG. JUDGE