IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

EMC CORPORATION,	)	
	)	
Plaintiff,	)	
	)	
v.	)	C.A. No. _____
	)	
PURE STORAGE, INC.,	)	**DEMAND FOR JURY TRIAL**
	)	
Defendant.	)	

COMPLAINT FOR INJUNCTIVE RELIEF AND DAMAGES FOR PATENT INFRINGEMENT

Plaintiff EMC Corporation ("EMC") alleges as follows against Defendant Pure Storage, Inc. ("Pure Storage" or "Defendant").

NATURE OF ACTION

1. This action arises from—and is necessary to remedy—Pure Storage's continued and willful infringement of EMC's patented technology. Pure Storage infringes EMC's United States Patent No. 7,434,015 ("the '015 patent"). The '015 patent covers critical technology used in large-scale data storage; specifically, the '015 patent relates to data deduplication, which Pure Storage has repeatedly and consistently described as mandatory, essential and critical to its products and its business. Data deduplication, which substantially reduces the data that needs to be stored, is so important to Pure Storage—its products and its business—that its senior executives have stated that Pure Storage would never remove that feature and functionality unless compelled to do so. For years, Pure Storage has incorporated data deduplication in its products that is covered by and infringes EMC's '015 patent.

2. On November 26, 2013, EMC sued Pure Storage, alleging that Pure Storage's FlashArray series 300 and 400 products infringed the '015 patent (C.A. No. 13-1985-RGA) ("the

2013 Action"). In the 2013 Action, Pure Storage was found to infringe the '015 patent, and the '015 patent also was found to be valid (despite Pure Storage's effort to invalidate it). In the 2013 Action, a jury awarded EMC $14 Million in damages for Pure Storage's infringement of the '015 patent. That damages award covered only a limited period of time. Specifically, the damages award was only for sales of Pure Storage products from November 2013 to January 2016. In addition, the damages award covered only a limited set of Pure Storage products. Specifically, the damages award was only for sales of Pure Storage's FlashArray series 300 and 400 products. On March 17, 2016, after a jury verdict, the Court entered judgment in favor EMC on the '015 patent in the 2013 Action.

3. Thus, although significant, the damages award in the 2013 Action was based on only a limited portion of sales of the FlashArray 300 and 400 models between November of 2013 and January of 2016. Additional Pure Storage products were not included in the 2013 Action. The 2013 Action did not address—and the damages award did not cover—the extensive sales of more recent Pure Storage products, including, but not limited to, Pure's current flagship "FlashArray//m" product. On information and belief, FlashArray//m sales already exceed in volume the sales of the FlashArray 300 and 400 models already found to infringe the '015 patent.

4. All versions of Pure Storage's FlashArray have included the deduplication technology found to infringe in the 2013 Action. Pure Storage's FlashArray//m, for example, infringes the '015 patent. In fact, the FlashArray//m product contains deduplication technology that is materially the same as the technology already found to infringe the '015 patent.[1]

[1] The FlashArray//m product was released after the complaint was filed in the 2013 Action and late in the discovery process. Accordingly, Pure Storage objected to its inclusion in

Accordingly, with its past and continued sales of FlashArray//m, Pure Storage has infringed and continues to infringe the '015 patent, and Pure Storage has no defense to liability. At a minimum, EMC is entitled to significant compensatory damages for these sales. Moreover, Pure Storage's infringement has been willful and deliberate, and EMC is entitled to additional remedies, including enhanced damages and attorneys' fees.

5. Despite the fact that all of the Pure Storage products at issue in the 2013 Pure Storage have been found to infringe the '015 patent, Pure Storage has announced that it intends to "continue to sell" infringing models of FlashArray, including FlashArray//m. *See* Exhibit A (http://blog.purestorage.com/litigation-update/ (accessed March 21, 2016)). Pure Storage has stated that it believes that it has alternative deduplication functionality that it may implement to try to avoid "ongoing royalties." As Pure Storage knows, however, its supposed alternative designs still infringe the '015 patent.

PARTIES

6. Plaintiff EMC is a Massachusetts corporation with its principal executive offices in Hopkinton, Massachusetts. EMC is a recognized leader in the information technology industry, offering innovative products and services that enable its customers to store, manage, protect, and analyze vast amounts of digital data in a trusted and cost-efficient way. EMC's extensive product offerings are used by customers around the world. Among the products EMC offers to the industry are data storage systems based on flash memory.

---

that action, and EMC reserved its rights to seek its remedies regarding the product in a subsequent case.

7. Defendant Pure Storage is a Delaware corporation with headquarters in Mountain View, California. Pure Storage manufactures and sells data storage systems based on flash memory, in competition with EMC, under names such as "FlashArray."

JURISDICTION AND VENUE

8. This is an action for patent infringement under the patent laws of the United States, Title 35 of the United States Code. This Court has subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338.

9. The Court has personal jurisdiction over Pure Storage because Pure Storage is incorporated in Delaware and has conducted and continues to conduct business within this judicial district.

10. Venue is proper in this judicial district under 28 U.S.C. §§ 1391(b) and (c) and 1400(b).

FACTUAL BACKGROUND

11. On October 7, 2008, the U.S. Patent and Trademark Office issued the '015 patent, entitled "Efficient Data Storage System." Ming Benjamin Zhu, Kai Li, and R. Hugo Patterson, who performed work for a company called Data Domain, Inc., invented the subject matter of the '015 patent, which relates to a novel method, device, and computer program product for deduplicating data. EMC, through its acquisition of Data Domain, obtained title and substantial rights in the '015 patent, including the right to bring this suit for injunctive relief and damages. *See* D.I. 430, *EMC Corp. v. Pure Storage, Inc.*, C.A. No. 13-1985-RGA (D. Del. Feb. 29, 2016)

12. In the 2013 Action, EMC accused data deduplication features in FlashArray's Purity Operating System, and the practice of methods relating to the use of those products, as meeting the claim limitations of the asserted claims of the '015 patent. On February 11, 2016,

the Court found as a matter of law that the accused FlashArray products directly infringed claims 1, 7, and 16 of the '015 patent. D.I. 381, *EMC Corp. v. Pure Storage, Inc.*, C.A. No. 13-1985-RGA (D. Del. Feb. 11, 2016). Pure Storage subsequently stipulated to infringement of claims 2 and 15 of the '015 patent, and further stipulated to inducement of infringement of all five asserted claims.

13. In the 2013 Action, the jury rejected Pure Storage's invalidity defenses, and the '015 patent was held valid.

14. During trial in the 2013 Action, Pure Storage alleged that it had developed—but not yet commercially released—certain purported non-infringing alternatives to the '015 patent. Pure Storage has also stated publicly that it "do[es] not expect to pay any ongoing royalties to EMC" because it has "alternatives ready to go for the software feature that the Court found to be infringing EMC's ['015 patent]." *See* Exhibit A (http://blog.purestorage.com/litigation-update/ (accessed March 16, 2016)). As Pure Storage knows—and as EMC showed at trial—Pure Storage's purported alternatives infringe the '015 patent.

15. In addition to continuing to sell the FlashArray products found as a matter of law to infringe the '015 patent, Pure Storage currently sells and offers to sell other products that infringe the '015 patent. For example, in 2015, Pure Storage released its FlashArray//m product. As explained in more detail below, on information and belief, all models of Pure Storage's FlashArray//m, including but not limited to //m10, //m20, //m50, and //m70, incorporate the same or similar features found to infringe the '015 patent. *See, e.g*, https://www.purestorage.com/content/dam/purestorage/pdf/datasheets/PureStorage_FlashArraym-Brochure.pdf at 5 ("always-on inline deduplication"); http://blog.purestorage.com/in-memory-database-meets-mighty-performance-storage-flasharraym-is-certified-for-sap-hana/

("FlashArray//m brings the following essentials to complete the promise of in-memory databases . . . both inline compression and deduplication"); https://www.purestorage.com/products/flash-array-m.html ("FlashArray//m is powered by software natively built to capitalize on flash: Purity Operating Environment is the software heart of the FlashArray").

CLAIM FOR RELIEF
Infringement of U.S. Patent No. 7,434,015

16. EMC hereby realleges and incorporates herein the allegations set forth in Paragraphs 1-13 of this Complaint.

17. EMC has title and substantial rights in the '015 patent, including the right to bring this suit for injunctive relief and damages. A copy of the '015 patent is attached hereto as Exhibit B.

18. EMC is informed and believes, and on that basis alleges, that Pure Storage has been and is directly infringing the '015 patent under at least 35 U.S.C § 271(a) by making, using, selling, and/or offering for sale systems, methods, and/or products that incorporate the deduplication inventions claimed in the '015 patent, including, but not limited to, by making, using, selling, and/or offering to sell FlashArray//m, in addition to other Pure Storage products introduced subsequent to the FlashArray 300 and 400 models that were accused in the 2013 Action, sold under other names, that contain the same or similar infringing deduplication functionality. EMC also alleges that Pure Storage directly infringes and will directly infringe the '015 patent by making and using code relating to Pure Storage's purported alternative ways of performing deduplication, including, but not limited to, the purported alternatives that Pure identified in the 2013 Action, including at trial. The deduplication products, code, features, and methods accused of infringing the '015 patent are hereinafter referred to as the Accused Instrumentalities.

19. EMC is informed and believes, and on that basis alleges, that Pure Storage's FlashArray//m product includes the same or substantially the same deduplication technology as the FlashArray 300 and 400 models that were found to infringe in the 2013 Action.

20. EMC is informed and believes, and on that basis alleges, that Pure Storage's infringement is literal or, in the alternative, that Pure Storage infringes under the doctrine of equivalents.

21. The '015 patent claims methods, devices, and computer program products for storing data. EMC is informed and believes, and on that basis alleges, that Pure Storage infringes at least claims 1, 2, 7, 15, and 16 of the '015 patent. For example, claim 1 recites:

> 1. A method for storing data comprising:
>
> [a] receiving a data stream comprising a plurality of data segments;
>
> [b] assigning an identifier to one of the plurality of data segments; and
>
> [c] determining whether one of the plurality of data segments has been stored previously using a summary, wherein the summary is a space efficient, probabilistic summary of segment information.

22. Upon information and belief, for example and without limitation, the Accused Instrumentalities practice the limitations of claim 1 at least by:

a. receiving a stream of data sectors from one or more sources;

b. assigning a hash value to one or more of the received data sectors;

c. determining whether the received data sector has been stored previously using, *inter alia*, a "Successful Dedupe" table, a "Recent" table, or other data structure that records the hash values of data sectors that have been previously stored.

23. The Accused Instrumentalities practice the limitations of other claims of the '015 patent for the same or similar reasons.

24. EMC is informed and believes, and on that basis alleges, that Pure Storage has induced infringement, at least because it instructs its customers on the use of its products, and that Pure Storage has continued such instruction with knowledge of the ’015 patent and the infringement of that patent, and with the intent to cause infringement, thereby continuing to induce infringement of the ’015 patent. Such instruction is provided, for example, in user guides, installation and support guides, and other documentation; in marketing videos; and through Pure Storage’s support services. Pure Storage has represented to customers that deduplication is “essential.” (http://blog.purestorage.com/reflections-on-a-puritan-new-year/). Pure Storage has caused its customers to use the Accused Instrumentalities in a manner that infringes the ’015 patent.

25. EMC is informed and believes, and on that basis alleges, that Pure Storage has contributed and is continuing to contribute to the infringement of the ’015 patent by selling or offering to sell the Accused Instrumentalities to its customers with knowledge that those products are especially made or especially adapted for use in the infringement of the ’015 patent. Those products, which are not staple articles of commerce suitable for substantial noninfringing uses, constitute at a minimum apparatuses for use in practicing a patented process of the ’015 patent.

26. EMC alleges that since at least the filing of the Complaint in the 2013 Action, Pure Storage’s infringement of the ’015 patent has been willful and deliberate. Pure has known about the ’015 patent since November of 2013 and has, at a minimum, acted with reckless disregard of a substantial risk of infringement of EMC’s valid patent.

27. Pure Storage’s infringement has left EMC with no adequate remedy at law and has caused, is causing, and if not enjoined will continue to cause irreparable damage to EMC.

28. Pure Storage, by way of its infringing activity, has caused and continues to cause EMC to suffer damages in an amount to be determined at trial.

PRAYER FOR RELIEF

Wherefore, EMC respectfully requests that judgment be entered in its favor and prays that the Court grant the following relief:

A. A judgment in favor of EMC that Pure Storage has infringed, directly and indirectly, literally and/or under the doctrine of equivalents, claims of the ’015 patent;

B. A judgment declaring that Pure Storage’s infringement of the ’015 patent has been willful and deliberate;

C. An order preliminarily and permanently enjoining Pure Storage, together with its officers, agents, employees, attorneys, dealers, distributors, sales representatives, and all others acting in concert or privity with it, from making, using, selling, offering for sale, or importing the Accused Instrumentalities, or any colorable imitation thereof, and from otherwise infringing the claims of the ’015 patent;

D. An order requiring Pure Storage to provide a pre-judgment accounting and to pay supplemental damages to EMC, including without limitation, pre-judgment and post- judgment interest;

E. An award to EMC of the damages, including enhanced damages, to which EMC is entitled under 35 U.S.C. § 284 for Pure Storage’s past infringement and any continuing or future infringement up until the date Pure Storage is finally and permanently enjoined from further infringement;

F. An award to EMC of equitable relief requiring Pure Storage to destroy all infringing products in inventory, including but not limited to the Accused Products wherever

they may be stored or maintained, and to recall from the marketplace all such infringing products, including but not limited to any infringing products in the possession or control of dealers, distributors, or customers;

G. An award to EMC of its attorneys' fees and costs in this action, including on the basis that this is an exceptional case under 35 U.S.C. § 285;

H. Such other relief that the Court deems just and proper.

JURY DEMAND

EMC demands a trial by jury on all issues triable to a jury.

MORRIS, NICHOLS, ARSHT & TUNNELL LLP

/s/ Jack B. Blumenfeld

Jack B. Blumenfeld (#1014)
Jeremy A. Tigan (#5239)
1201 North Market Street
P.O. Box 1347
Wilmington, DE 19899
(302) 658-9200
jblumenfeld@mnat.com
jtigan@mnat.com

Attorneys for Plaintiff EMC Corporation

OF COUNSEL:

Josh A. Krevitt
Paul E. Torchia
GIBSON, DUNN & CRUTCHER LLP
200 Park Avenue
New York, NY 10166-0193
(212) 351-2490

Stuart M. Rosenberg
GIBSON, DUNN & CRUTCHER LLP
1881 Page Mill Road
Palo Alto, CA 94304-1211
(650) 849-5389

Paul T. Dacier
Krishnendu Gupta
William R. Clark
Thomas A. Brown
EMC CORPORATION
176 South Street
Hopkinton, MA 01748

March 21, 2016

EXHIBIT A

Litigation Update

03.15.2016 Posted by Joe FitzGerald

Posted In: Company, Competition

In November 2013, EMC sued Pure in federal court in Delaware, claiming that Pure infringed five patents involving various data storage technologies. Last week, the case went to trial – and the case wrapped up this afternoon.

Before the trial started, EMC dropped one patent from the case, and the judge in a pre-trial summary judgment ruling found that we didn't infringe another. In the same ruling, the judge hearing the case found that Pure infringed certain claims of one of EMC's patents related to de-duplication technology.

The seven-day trial, therefore, focused on two questions:

- Whether we infringed two EMC patents dealing with de-duplication and RAID storage.
- Whether the de-duplication patent that the judge found we infringed was actually valid.

The jury's verdict was split.

- The jury found that Pure did not infringe either of the two patents they were asked to consider.
- However, the jury also found to be valid the one patent that we were found to have infringed by the judge.

The jury awarded EMC damages of $14 million. EMC had originally sought more than $80 million.

Our view has been and remains that EMC's litigious approach to competition primarily reflects efforts to stabilize its storage business as customers around the world abandon the kind of disk-based storage systems EMC pioneered in favor of flash-based storage from innovative companies like Pure. As the trial proceedings made clear, EMC built its own flash-based storage products via acquisition, rather than organic innovation.

We are gratified that the jury agreed with our view of the facts on most of the issues at trial, although we are disappointed with the one ruling not in our favor on one of EMC's de-duplication patents. We continue to believe that both the facts and the law are on our side on that issue – and we are considering our options for appealing that aspect of the decision.

It is important to note this ruling will not disrupt Pure, our customers or our partners:

- We will continue to sell the same set of products and services that drove 150% top-line growth in our fiscal year ended January 31, 2016.
- We do not expect to pay any ongoing royalties to EMC.
- We have alternatives ready to go for the software feature that the Court found to be infringing EMC's de-dupe patent.
- Pure has sufficient financial resources such that the proposed damages – if upheld on appeal – will not slow us down.

As Pure CEO Scott Dietzen wrote in a blog post in 2013, Pure welcomes marketplace competition. Competition drives innovation and customer value. Competition makes our products better and makes us into a better company, more attuned to customer and partner needs. Competition also fuels market growth. We look forward to continuing to compete with EMC in the public marketplace.

EXHIBIT B

US007434015B2

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

(10) **Patent No.: US 7,434,015 B2**
(45) **Date of Patent: Oct. 7, 2008**

(54) **EFFICIENT DATA STORAGE SYSTEM**

(76) Inventors: **Ming Benjamin Zhu**, 2929 Campus Dr., Suite 250, San Mateo, CA (US) 94403; **Kai Li**, 2929 Campus Dr., Suite 250, San Mateo, CA (US) 94403; **R. Hugo Patterson**, 2929 Campus Dr., Suite 250, San Mateo, CA (US) 94403

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **11/974,961**

(22) Filed: **Oct. 16, 2007**

(65) **Prior Publication Data**

US 2008/0133835 A1 Jun. 5, 2008

Related U.S. Application Data

(63) Continuation of application No. 11/403,153, filed on Apr. 11, 2006, now Pat. No. 7,305,532, which is a continuation of application No. 10/325,479, filed on Dec. 20, 2002, now Pat. No. 7,065,619.

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

(52) **U.S. Cl.** **711/162**; 711/118

(58) **Field of Classification Search** 711/162, 711/118
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,765,173	A *	6/1998	Cane et al.	707/204
6,269,431	B1 *	7/2001	Dunham	711/162
6,366,987	B1 *	4/2002	Tzelnic et al.	711/162
6,449,697	B1 *	9/2002	Beardsley et al.	711/137
6,549,992	B1 *	4/2003	Armangau et al.	711/162
6,754,765	B1 *	6/2004	Chang et al.	711/103
2003/0093647	A1 *	5/2003	Mogi et al.	712/1
2003/0110411	A1 *	6/2003	Harari et al.	714/8
2003/0204605	A1 *	10/2003	Hudson et al.	709/228

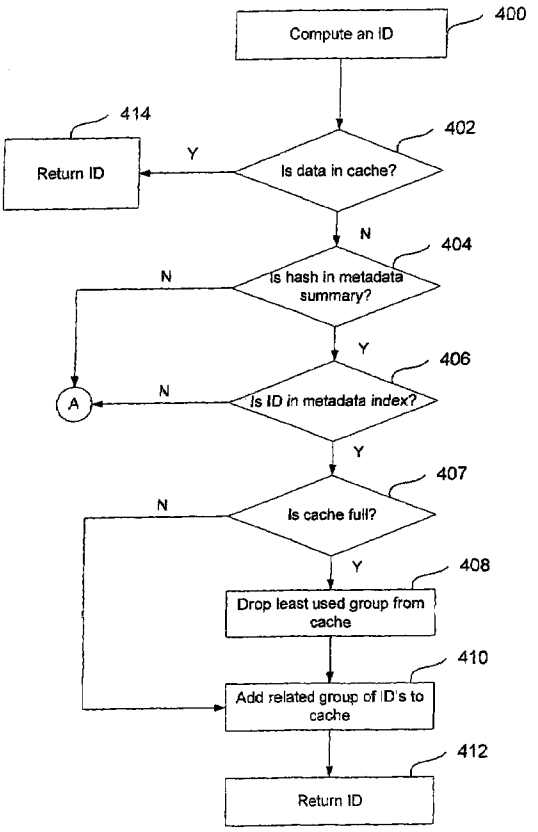
* cited by examiner

Primary Examiner—Brian R Peugh
(74) *Attorney, Agent, or Firm*—Van Pelt, Yi & James LLP

(57) **ABSTRACT**

A system and method are disclosed for providing efficient data storage. A data stream comprising a plurality of data segments is received. The system determines whether one of the plurality of data segments has been stored previously using a summary in a low latency memory; in the event that the data segment is determined not to have been stored previously, assigning an identifier to the data segment.

16 Claims, 7 Drawing Sheets



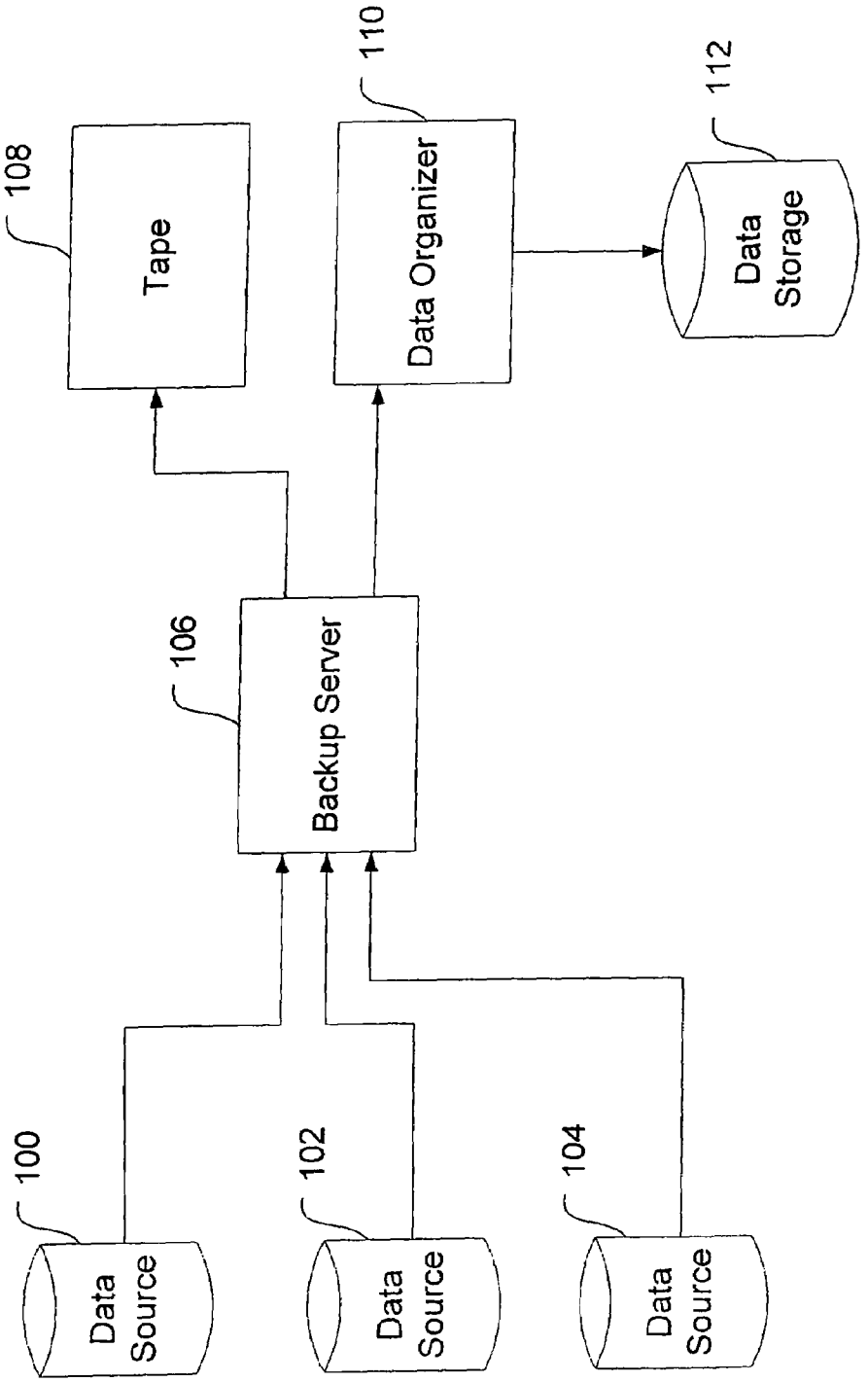


Fig. 1
PRIOR ART

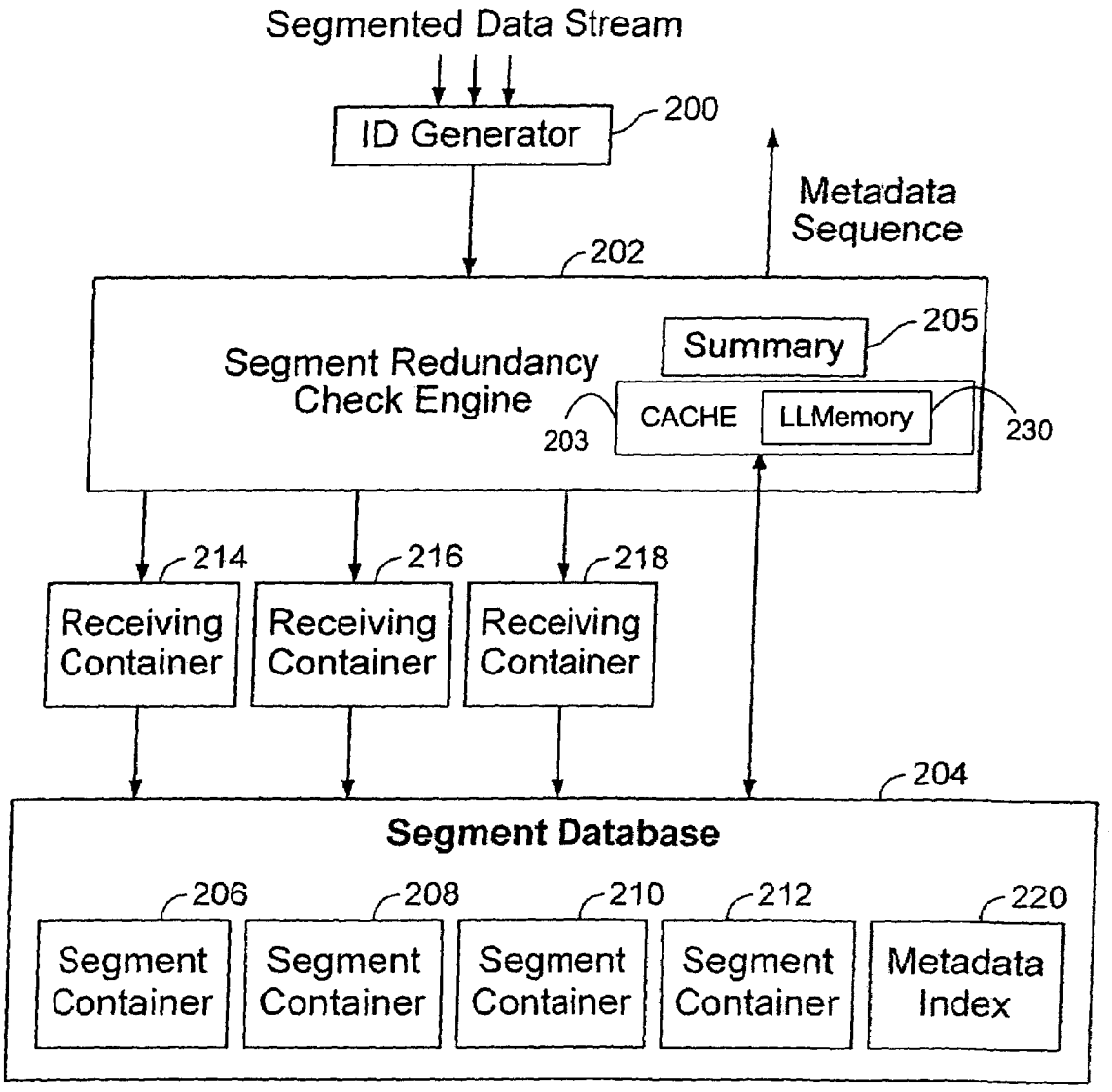

Segmented Data Stream
ID Generator
200
Metadata Sequence
202
Segment Redundancy Check Engine
Summary
205
CACHE
LLMemory
203
230
214
216
218
Receiving Container
Receiving Container
Receiving Container
204
Segment Database
206
208
210
212
220
Segment Container
Segment Container
Segment Container
Segment Container
Metadata Index

FIG. 2

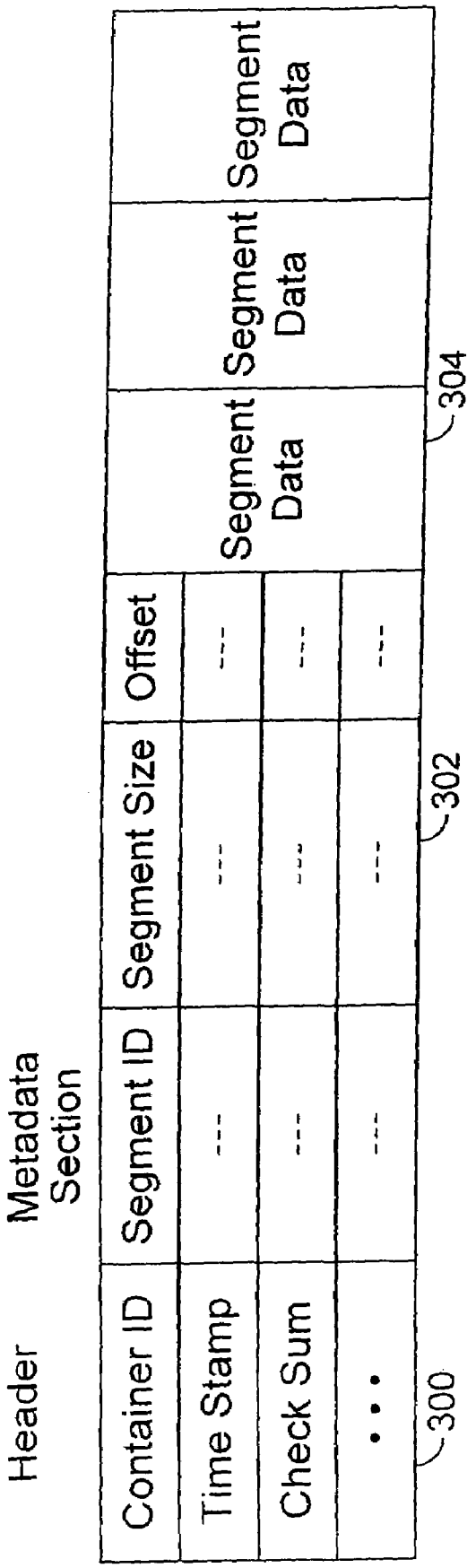

Header
Metadata Section
Container ID
Time Stamp
Check Sum
• • •
Segment ID
Segment Size
Offset
Segment Data
Segment Data
Segment Data
300
302
304

FIG. 3

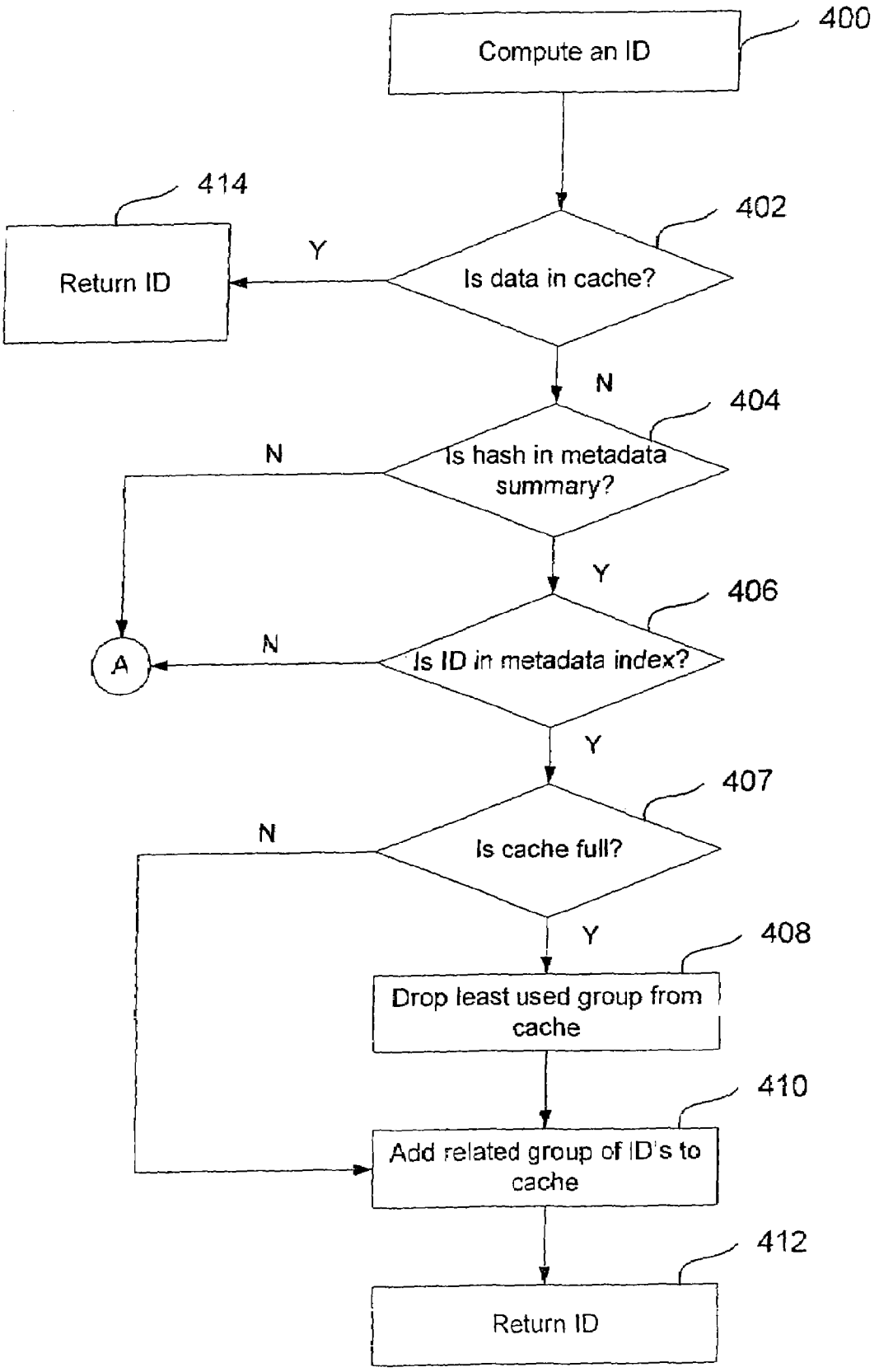

400
Compute an ID
414
402
Return ID
Y
Is data in cache?
N
404
N
Is hash in metadata summary?
Y
406
A
N
Is ID in metadata index?
Y
407
N
Is cache full?
Y
408
Drop least used group from cache
410
Add related group of ID's to cache
412
Return ID

Fig. 4A

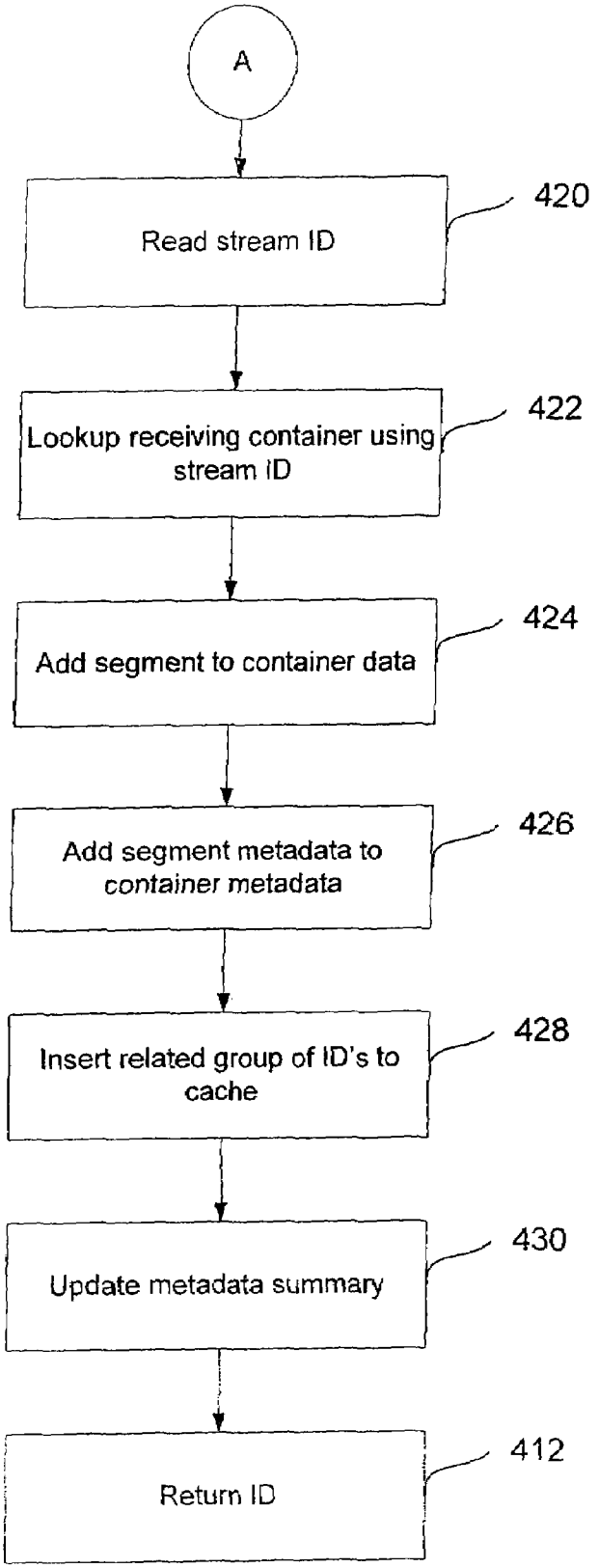

A
Read stream ID
420
Lookup receiving container using stream ID
422
Add segment to container data
424
Add segment metadata to container metadata
426
Insert related group of ID's to cache
428
Update metadata summary
430
Return ID
412

Fig. 4B

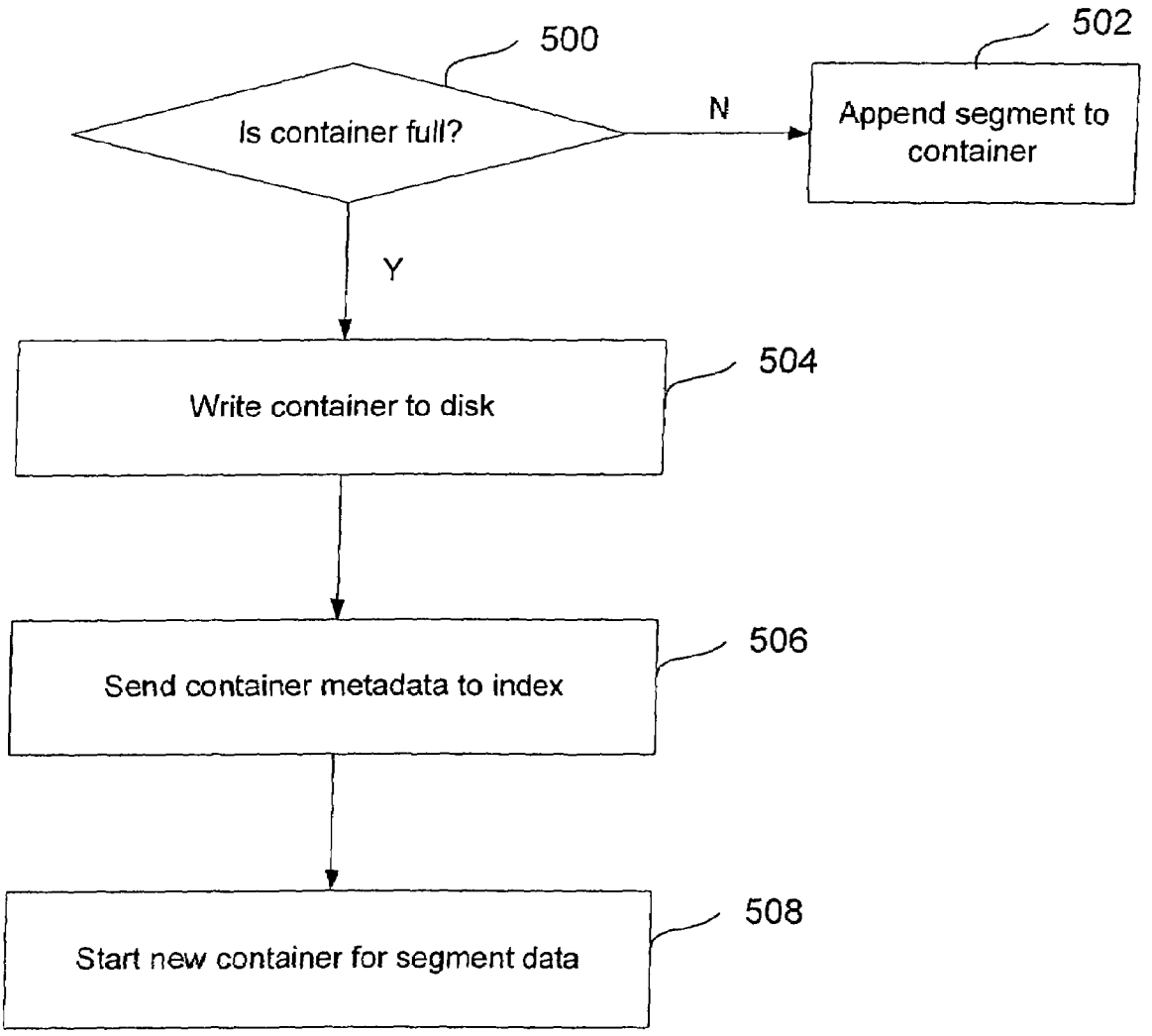

500
Is container full?
N
502
Append segment to container
Y
Write container to disk
504
Send container metadata to index
506
Start new container for segment data
508

Fig. 5

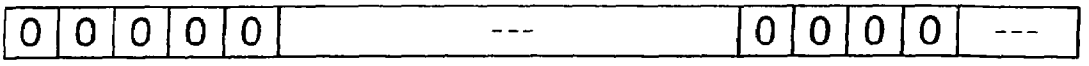


FIG. 6A

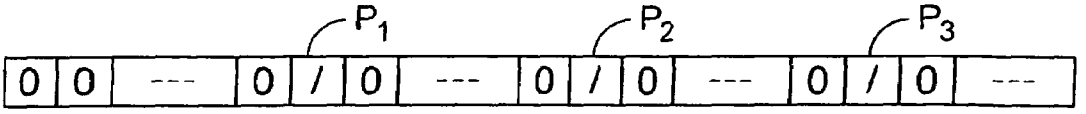


FIG. 6B

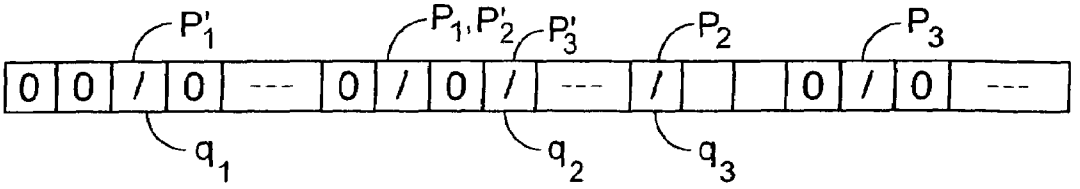


FIG. 6C

EFFICIENT DATA STORAGE SYSTEM

CROSS REFERENCE TO RELATED APPLICATIONS

This application is a continuation of U.S. patent application Ser. No. 11/403,153, entitled EFFICIENT DATA STORAGE SYSTEM filed Apr. 11, 2006, now U.S. Pat. No. 7,305,532 which is incorporated herein by reference for all purposes, which is a continuation of Issued U.S. patent application Ser. No. 10/325,479, entitled EFFICIENT DATA STORAGE SYSTEM filed Dec. 20, 2002, now U.S. Pat. No. 7,065,619, which is incorporated herein by reference for all purposes.

FIELD OF THE INVENTION

The present invention relates generally to data storage systems. More specifically, a data storage system that efficiently eliminates redundancy is disclosed.

BACKGROUND OF THE INVENTION

Enterprises as well as individuals are becoming increasingly dependent on computers. As more and more data are generated, the need for efficient and reliable data backup storage systems is increasing. There are a variety of systems in existence today, utilizing both local and network storage for backup.

FIG. **1** is a block diagram illustrating a typical network backup system. Data are generated from a variety of sources, for instance data sources **100**, **102** and **104**. During the backup operation, the data sources stream their data contents to backup server **106**. The backup server receives the data streams, optionally processes the data streams, and sends the data to backup devices such as tape **108** and data organizer **110**. Data organizer **110** processes the data received and writes the data to a storage device **112**, which can be a single disk or a disk array. The data organizer can be a device separate from the backup server or a part of the backup server.

During a backup operation, the data from the data sources are copied to the backup devices. Commonly, there is a substantial amount of data from each of the data sources that remains the same between two consecutive backups, and sometimes there are several copies of the same data. Thus, the system would be more efficient if unchanged data are not replicated.

There have been attempts to prevent redundant copying of data that stay the same between backups. One approach is to divide the data streams from the data sources into segments and store the segments in a hash table on disk. During subsequent backup operations, the data streams are again segmented and the segments are looked up in the hash table to determine whether a data segment was already stored previously. If an identical segment is found, the data segment is not stored again; otherwise, the new data segment is stored. Other alternative approaches including storing the segments in a binary tree and determining whether an incoming segment should be stored by searching in the binary tree.

While such an approach achieves some efficiency gains by not copying the same data twice, it incurs significant disk input/output (I/O) overhead as a result of constantly accessing the disk to search for the data segments. Also, the searching techniques employed in the existing systems often involve searching for the ID in a database, which becomes less efficient as the size of the database grows. It would be desirable to have a backup system that would reduce the disk I/O overhead and increase search efficiency, while eliminating the unnecessary data replication.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

FIG. **1** is a block diagram illustrating a typical network backup system.

FIG. **2** is a block diagram illustrating a storage system embodiment according to the present invention.

FIG. **3** illustrates the data layout of a container embodiment according to the present invention.

FIG. **4**A and FIG. **4**B are flowcharts illustrating the handling of an incoming segment in a storage system embodiment in accordance with the present invention.

FIG. **5** is a flowchart illustrating the details of adding a new segment to the receiving container step shown in FIG. **4**B.

FIG. **6**A-FIG. **6**C illustrate the operations of a Bloom filter.

DETAILED DESCRIPTION

It should be appreciated that the present invention can be implemented in numerous ways, including as a process, an apparatus, a system, or a computer readable medium such as a computer readable storage medium or a computer network wherein program instructions are sent over optical or electronic communication links. It should be noted that the order of the steps of disclosed processes may be altered within the scope of the invention.

A detailed description of one or more preferred embodiments of the invention is provided below along with accompanying figures that illustrate by way of example the principles of the invention. While the invention is described in connection with such embodiments, it should be understood that the invention is not limited to any embodiment. On the contrary, the scope of the invention is limited only by the appended claims and the invention encompasses numerous alternatives, modifications and equivalents. For the purpose of example, numerous specific details are set forth in the following description in order to provide a thorough understanding of the present invention. The present invention may be practiced according to the claims without some or all of these specific details. For the purpose of clarity, technical material that is known in the technical fields related to the invention has not been described in detail so that the present invention is not unnecessarily obscured.

An improved storage system that eliminates redundant copying of identical data during a backup operation is disclosed. The system receives a segmented input data stream and produces segment ID's for the segments. Checks are performed on the data segments to determine whether the same segments have previously been stored to a segment database of the system, thereby avoiding redundant copying. Preliminary checking techniques are used to lower the latency associated with the checking and increase search efficiency. In one embodiment, metadata information about segments that are likely to be encountered soon are stored in a metadata cache and used in the preliminary check. In one embodiment, a summary is used in the preliminary check. In some embodiments, the cache and summary techniques are combined.

FIG. **2** is a block diagram illustrating a storage system embodiment according to the present invention. One or more data streams from a backup server or other data source are divided into segments (also referred to as blocks), and the

segmented data streams are received by an ID generator **200**. The size of the segments varies depending on the implementation. In some embodiments, the segments have a fixed size. In some embodiments, the segments have variable sizes. In some embodiments, the data stream is broken into a number of parallel streams where the streams may have different segment sizes.

If the data stream is not segmented at the source of the data stream, then the stream may be separated into segments such that the segments can be readily matched with segments from previous or future streams according to the techniques disclosed in Finding Similar Files in A Large File System (Udi Manber, Technical Report TR 93-33, University of Arizona, October 1993.)

A segment ID is generated by ID generator **200** and assigned to each of the segments received. The location of the ID generator is implementation dependent. In the embodiment shown, the IDs are generated before the segments are sent to the segment redundancy check engine. In some embodiments, the IDs are generated sometime after the segments have been processed by the segment redundancy check engine. In certain embodiments, the IDs are generated when the segments are ready to be stored to segment database **204**. The methods used to generate the ID are also implementation dependent. In some embodiments, the ID is the segment data itself. In some embodiments, a digital signature (also referred to as a cryptographic hash or a fingerprint), is generated from the result of a hash function computed using the segment data. In some embodiments, a cryptographic hash function such as the MD5 algorithm is used. In one embodiment, the ID is a Rabin fingerprint. In some embodiments, the ID is a sequential number assigned by the system.

In this embodiment, the segment data stream sent to segment redundancy check engine **202** includes both the segment data and the segment IDs. In other embodiments, the segment IDs are not sent to the segment redundancy check engine. The segment redundancy check engine is designed to efficiently determine whether segments are already stored by the system while reducing latency. The segment redundancy check engine reduces the amount of time required for most redundancy checks by performing certain preliminary checks to determine whether the segment has been stored previously, using operations that are carried out in quickly accessible memory.

Segment redundancy check engine **202** accesses a cache **203** that stores segment information for fast preliminary checks. In various embodiments, the segment information includes segment ID's, segment metadata, segment data, or combinations thereof. Cache **203** is typically implemented using memory that is quickly accessible, such as various kinds of dynamic random access memory, as well as various forms of non-volatile memory. Such memory or any other similarly quickly accessible memory is referred to as low latency memory. In general, low latency memory is any type of memory or cache that can generally be read more quickly or has better throughput than the large memory that stores the entire segment database. In the embodiment shown, the segment redundancy check engine also accesses a summary **205** that is implemented in memory, used to determine whether a segment has been stored previously.

If the preliminary checks do not conclusively determine whether the segment has already been stored, then a lookup is done in segment database **204** to confirm whether the segment has been stored previously. Segment database **204** is typically stored in a relatively high latency memory. A relatively high latency memory refers to various types of storage that cannot be addressed as quickly as the quickly accessible memory of the system, for example, hard disk, optical storage, devices over a network, etc. There are different causes for a storage to have high latency. For example, the storage has a small throughput due to the bus speed of its interface; or the storage is large in size and thus accessing specific items involves searching a large amount of data; or the storage is connected to the rest of the system via a network; or the storage is accessed often and a queue may develop or other problems may occur.

For the purpose of example, the segment databases in embodiments discussed in the rest of this specification are stored on hard disk, although it should be understood that other types of high latency memory can be used. Data segments and their associated metadata are stored in segment database **204**. The segment database is content addressable, which is to say that given the content of a data segment, a lookup can be done in the segment database to confirm whether the segment has been stored previously. In the embodiment shown, segment database **204** includes a segment metadata index and multiple segment containers **206-212** that each stores multiple segments along with segment metadata. The segment metadata index provides a way to quickly look up the storage location of a segment. In different embodiments, the metadata index may be implemented as a hash table, a tree, a list, a combination thereof, etc. Only a small number of segment containers are illustrated for the purpose of example; an actual system would have a large number of such containers. In the embodiment shown, there are a number of receiving containers **214-218** that reside in memory and serve as buffers to store the newly received segments before they are written to segment containers on disk. Again, the number of receiving containers are different in different embodiments, although in many cases there is one receiving container per segment stream.

The containers are the basic storage units used in the segment database. A container is a relatively large chunk of data storage space (as much as 8 MB or more in some embodiments) used mainly to store data segments and segment metadata. In the embodiment shown, two types of containers, receiving and storage, are used. Each receiving container is used to store data segments and their metadata received from a single data source. Data segments from the same data source are added in the corresponding receiving container sequentially, and the metadata section of the receiving container is updated accordingly. The receiving containers are kept in memory so that new segments can be efficiently processed.

In this embodiment, once a receiving container is filled with data, it is written to disk as a whole. The storage units for containers on disk in the segment database are referred to as segment containers. A segment container may be read in its entirety, one section at a time, or in byte ranges within a section. The disk I/O performance is improved by using receiving containers to buffer data read from the stream and segment containers to write data to the segment database in large chunks. In certain embodiments, there is a container manager that is responsible for functions such as allocating, deleting, reading, writing and reliably storing the containers. The size of the containers are the same in the embodiment shown; they are different sizes in other embodiments. Generally, the receiving container and the segment container use the same data format. FIG. **3** illustrates the data layout of a container embodiment according to the present invention. **300** is the header portion of the container, which includes information related to the container such as container Id, time stamp, checksum, error correction codes, etc. **304** is the data section that stores the segment data. In some embodiments, the data segments are stored in compressed form. In one

embodiment, a variation of Ziv-Lempel compression algorithm is used. In other embodiments, different compression techniques are applied to the data segments. **302** is the metadata section that stores the metadata associated with the corresponding segment data, such as the segment ID, segment size, and offset from the start of the container so that the segment can be accessed. In various embodiments, the metadata section may be implemented as an array, a list, a tree, a table, etc.

Returning to FIG. **2**, metadata index **220** is used to confirm whether the data segment has already been stored. The metadata stored in the index is content dependent, in other words, the metadata are generated based on the content of a data segment and can be used to locate a data segment in the database if the data segment has been stored previously. In different embodiments, different types of metadata are used in the cache. In the embodiment shown, the metadata index is implemented as a hash table that is organized in buckets. The buckets store the segment ID's and other associated metadata, such as the ID of the container that stores the segment. To add new segment information, a hash function is applied to the segment ID to determine in which bucket the ID should be stored, and then the segment ID and its associated information is stored to the bucket. To look up a segment ID, the segment ID is hashed and the bucket corresponding to the hash value is located. A search is performed on all the segment ID's stored in the bucket to determine whether the segment ID already exists.

Typically, a storage system will store a very large number of segments and therefore a very large number of segment ID's are stored in the metadata index. Cache **203** stores only a small subset of the segment for fast preliminary determination of whether a received segment has already been stored. The segment redundancy check engine checks the complete metadata index stored on disk to confirm whether the data has been stored previously, if a segment cannot be found in cache **203** or if the result of the summary check is inconclusive.

The nature of the determination that is made from checking the cache and the summary should be noted. The cache can positively (that is, conclusively) determine that the segment is has previously been stored, because it is found in the cache. If the segment is not in the cache, then there is no positive determination that the segment is not in the larger high latency database that holds all the segments. The summary can positively determine that the segment has not been stored. However, if the summary includes the segment, that is not a positive determination that the segment has been stored, since other segments or combinations of other segments can cause false hits in the summary. So, after checking the cache and the summary, the result is one of three possibilities:

1. The summary positively determines that the segment is new.
2. The cache positively determines that the segment was previously stored.
3. Neither the summary nor the cache makes a positive determination.

If neither the summary nor the cache makes a positive determination, then the larger high latency database must be searched to make a positive determination.

Different types of segment information are stored in cache **203** in various embodiments, including segment ID's, segment data, segment metadata, or combinations thereof. In some embodiments, the segment containers are cached and used in the preliminary check. In some embodiments, the metadata information of the segments are cached, and the preliminary checking is performed using the metadata information. For instance, the metadata information may include a short ID assigned to a data segment and a signature of the data segment, where the short ID is not likely to be unique. The preliminary check then involves looking up the short ID, and then the signature of a segment in the cache to determine whether the segment has been stored previously. Embodiments using segment ID in the cache are discussed for the purpose of example hereafter; it should be noted that other types of segment information can be used in the cache as well. The metadata are organized in the cache to allow for fast lookups. In various embodiments, the metadata may be stored in a hash table, a tree, a binary tree, a list, etc.

Accessing the segment metadata index on the hard disk is relatively expensive in terms of time. For that reason, it is important that the segment ID's of the segments most likely to be encountered in the data stream are in the cache and that space in the cache is not wasted on segment ID's that are not likely to be encountered.

In the embodiment shown, the groups of segment ID's that are transferred to and from the cache correspond to the groups of segments that are stored in segment containers. In other embodiments, segment ID's may be grouped independently of how the segments themselves are stored. To decrease the likelihood of cache misses, segment ID's are preferably grouped in some manner wherein when one segment ID in the group of segment ID's is encountered in the incoming data stream, other segment ID's in the group of segment ID's are likely to be encountered soon.

An important consideration in increasing the likelihood of cache hits is that the segments corresponding to each group of segment ID's be related, that is, that they generally are received closely together. In many systems, the incoming data stream may include segments from a number of sources that are interleaved. Data from a single source is likely to be related, but consecutive segments in an interleaved stream are not necessarily interleaved in the same manner if their respective sources are independent. In one embodiment, the sources provide stream with identifiers that are used to separate the interleaved stream into substreams corresponding to segments from a single source. Segments from each substream are stored in a different receiving container so that related segments are stored in the same place. In addition, the segment ID's for each receiving container are therefore related and comprise a useful group to be transferred to and from the cache together. Again it should be noted that the segment ID groups described in this example correspond to groups of segments stored in a container but in other embodiments, groups of segment ID's may be defined without any correspondence to how segments themselves are stored. Additionally, other related groups of segment information may be used instead of segment ID's.

In this embodiment, when the cache is full, the group of segment ID's that is the least recently used group of container segment ID's is dropped from the cache to make room for a new group of segment ID's from a newly accessed container. The next time segment ID's that are the same as the ones from the container that includes the dropped group of segment ID's are encountered in the data stream, the segment redundancy check engine will not find the segment ID's in the cache, and will then check the metadata index for the information. In some embodiments, the segment ID's can be dropped individually from the cache as opposed to being dropped in a group. In some embodiments, the cached segment ID's are stored in a first in first out (FIFO) queue, and the dropped segments are the segments that are stored to the queue the earliest.

In the embodiment shown, accessing segment ID's from a segment container will prompt the system to transfer all of the

segment ID's from that container to the cache, and the least recently used group of segment ID's will be dropped from the cache. In some embodiments, one or more segment ID's are dropped from the cache and the ID's are not grouped. The segment ID's in the updated cache are more likely to be related to the segment ID's of the incoming data stream, therefore cache hits are more likely and the I/O overhead associated with checking the metadata index is reduced. The system maintains a least recently used list that tracks when a segment ID's group was accessed. The list is used to replace the data that has not been accessed for the longest time with newly accessed data.

In some embodiments, the output of the segment redundancy check engine is a sequence of ID's that is stored and used later for reconstructing the data stream when the system performs a read operation. In various embodiments, the sequence of ID's may be segment ID's, a set of sequential numbers assigned to the data segments, or other types of segment metadata. The sequence of ID's can be used in combination with the segments stored in the segment containers to recreate the data stream. Thus, the ID sequence is a highly compressed representation of the incoming data stream that can be uncompressed by retrieving each of the segments in the segment database that are identified by the ID's. There are different ways to retrieve the segments using the ID sequence, such as using the ID information to look up the segments in the metadata cache, or finding the segment metadata in the metadata index and using the segment container and information in the metadata index to locate the segment itself.

FIG. **4**A and FIG. **4**B are flowcharts illustrating the handling of an incoming segment in a storage system embodiment in accordance with the present invention. In this embodiment, the segment ID's are stored in the cache. Thus, a segment ID is looked up in the cache to determine whether the segment has been stored previously. It should be noted that in other embodiments, other types of segment information can be stored in the cache for lookups instead of the segment ID's. At the beginning, a segment ID is generated for a data segment (**400**). In other embodiments, this step may be implemented elsewhere in the flowchart or omitted entirely. The segment redundancy check engine then performs a first preliminary check to determine whether the segment has been stored by looking up the segment ID in the engine's cache (**402**). This step should eliminate a good portion of data segments that are repeated. The cache stores the groups of selected segment information in memory, allowing for fast lookups of segment information as well as fast read operations.

If the segment ID is found in the metadata cache, the segment ID is returned to the segment redundancy check engine, and the segment is discarded (**414**). If, however, the segment ID is not found in the metadata cache, the segment redundancy check engine proceeds to perform a second preliminary check using a summary (**404**). A summary is a space efficient, probabilistic way of summarizing the segment database. It is designed to use a small amount of memory to summarize which segments are in the segment database. The details of the summary operations are described later in FIG. **6**. In this embodiment, a hash of the segment information is used to determine whether the segment is in the summary.

In this embodiment, when the hash is not found in the summary, it means that the segment does not exist in the segment database, the segment is new and control is transferred to point A in FIG. **4**B. On the other hand, if the hash is found in the summary, it does not necessarily mean that the segment data exists in the segment database. In other words, the summary accurately determines when the segment does not exist in the segment database and does not give false negatives; however, it may give false positives with some small probability. Thus, further action should be taken to ascertain whether the segment indeed exists in the segment database. In this embodiment, a confirmation step is used to positively determine whether the segment exists in the database. In the embodiment shown, looking up the segment ID in the metadata index confirms whether the ID and its corresponding data segment have already been stored (**406**).

If the ID is found in the metadata index, the cache is updated by reading a group of related metadata or segment data into the cache. If the cache is full (**407**), then the least recently used ID's or group of ID's are dropped from the cache (**408**). If the cache is not full, then control is transferred to (**410**). A related group of ID's are then added to the cache (**410**). The segment ID is returned to the segment redundancy check engine (**412**) and the segment data are discarded.

If the hash is not found in the summary, or if the ID is not found in the metadata index, then the data segment and the ID are new and various data structures in the system should be updated to reflect the addition of the new data. Control is transferred to point A in FIG. **4**B. A stream ID is extracted from the incoming data (FIG. **4**B, **420**), and the receiving container corresponding to the stream is located using the stream ID (**422**). The segment is then added to the receiving container (**424**). The segment metadata are added to the container's metadata section (**426**), and the group of related segment ID's are added to the cache if needed (**428**). The summary is also updated accordingly (**430**). The segment ID is returned to the segment redundancy check engine to be added to the segment ID sequence (**412**).

The preliminary checking steps provide ways to more efficiently determine whether a data segment has been stored previously. The checking in metadata cache and the checking in summary can be independent of each other. In some embodiments, step **404** occurs prior to step **402**. In certain embodiments, one of the two preliminary checking steps is implemented.

FIG. **5** is a flowchart illustrating the details of adding a new segment to the receiving container step (FIG. **4**B **424**). First, is determined whether the receiving container is full (**500**). If it is not full, the segment is appended to the container's data section (**502**). If it is full, the receiving container is written to disk (**504**) and its metadata are added to the metadata index (**506**). A new receiving container is then created to receive future segment data (**508**). The in-memory receiving container buffers the segment data and reduces the I/O overhead.

The summary used in FIG. **4**A step **404** is a space efficient, probabilistic summary of the database designed to use minimal amount of memory to summarize the data segments in the database. In one embodiment, the summary is generated from the segment ID's. In other embodiments, the summary is generated using the segment data. In certain embodiments, the summary is generated from other metadata associated with the segment.

In one embodiment, the summary is implemented using a summary vector. One example of such a summary vector is a "Bloom filter." FIG. **6**A-FIG. **6**C illustrate the operations of a Bloom filter. A Bloom filter uses a summary vector of m bits to summarize the information about n data items. The summary vector is updated when a new data item is received.

Initially, all the bits in the summary vector are set to 0, as shown in FIG. **6**A. A set of k independent hash functions h**1**, h**2**, . . . hk are applied to the segment. Different parameters of the segment can be used by the hash in different embodiments, including the data segment itself, parts of the data segment, metadata of the data segment, etc. In this embodi-

ment, a segment ID, a, is used; and the results are h1(a)=p1, h2(a)=p2, . . . hk(a)=pk, where p1-pk are numbers within a range between 1 to m. In the embodiment shown, k equals 3.

The bits at positions p1, p2, . . . pk are then set to 1 in the summary vector as shown in FIG. 6B, as a is added to the metadata index. Other inputs are hashed and the bits in the summary vector are set in a similar fashion. For instance, the hash functions are applied to a different ID, b, to obtain results h1(b)=p'1, h2(b)=p'2, . . . hk(b)=p'k. The bits at positions p'1, p'2, . . . p'k are set to 1 in the summary vector as shown in FIG. 6C. To determine whether an input ID x has already been added to the metadata index, first the hash functions are applied to x to obtain a new set of results h1(x)=q1, h2(x)=q2, . . . hk(x)=qk. Then, the bit positions in the summary vector that correspond to q1, q2, . . . qk are checked. If any of the bit positions is 0, then it is positively determined that x has never been updated in the summary vector and is not in the metadata index. If, however, all the bit positions are 1 as shown in FIG. 6C, it only indicates that x may already be in the database since the bit positions may have been set to 1 by a combination of two or more other IDs. Thus, further checking is needed to confirm whether the ID has indeed been stored previously. In some embodiments, the confirmation is performed by looking up the ID in the metadata index.

There are many applicable hash functions used in different embodiments, and the number of hash function used is implementation dependent. For example, the data bits of the ID can be divided into chunks and the results of the hash functions are the individual chunks. In one embodiment, a 160 bit long ID is divided into five chunks of 32 bits each by a set of 5 hash functions. In some embodiments, a group of $\log_2 m$ bits are selected from a single large hash. An improved system and method have been disclosed for efficiently storing data. The system receives a segmented input data stream and produces segment ID's. The system performs checks based on segment ID's to determine whether the same segments have previously been stored, thereby avoiding redundant copying. Preliminary checking techniques including caching and summary are used to efficiently determine the redundancy and minimize the latency associated with the checking.

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may be practiced within the scope of the appended claims. It should be noted that there are many alternative ways of implementing both the process and apparatus of the present invention. Accordingly, the present embodiments are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope and equivalents of the appended claims.

What is claimed is:

1. A method for storing data comprising:

receiving a data stream comprising a plurality of data segments;

assigning an identifier to one of the plurality of data segments; and

determining whether one of the plurality of data segments has been stored previously using a summary, wherein the summary is a space efficient, probabilistic summary of segment information.

2. A method for storing data as recited in claim 1 wherein the identifier is derived from the content of the one of the plurality of data segments.

3. A method for storing data as recited in claim 1 wherein the identifier is not derived from the content of the one of the plurality of data segments.

4. A method for storing data as recited in claim 1 wherein the identifier is a digital signature.

5. A method for storing data as recited in claim 1 further comprising storing the one of the plurality of data segments.

6. A method for storing data as recited in claim 1 wherein the determination can positively determine that the one of the plurality of data segments has not been stored previously, but cannot positively determine that the one of the plurality of data segments has been stored previously.

7. A method for storing data as recited in claim 1 further comprising:

confirming whether the one of the plurality of data segments has been stored previously using a relatively high latency memory.

8. A method for storing data as recited in claim 1 further comprising:

confirming whether the one of the plurality of data segments has been stored previously by checking in a cache.

9. A method for storing data as recited in claim 1 further comprising

confirming whether the one of the plurality of data segments has been stored previously, wherein confirming whether the data segment has been stored previously includes checking a cache; and

in the event that checking a cache results in a cache miss, confirming whether the one of the plurality of data segments has been stored previously further includes checking in a segment database.

10. A method for storing data as recited in claim 1 further comprising:

confirming whether the one of the plurality of data segments has been stored previously, wherein confirming whether the one of the plurality of data segments has been stored previously includes checking a cache; and

in the event that checking a cache results in a cache miss, confirming whether the one of the plurality of data segments has been stored previously further includes checking in a segment database, wherein the segment database is stored in relatively high latency memory.

11. A method for storing data as recited in claim 1 further comprising generating segment information for each of the one of the plurality of data segments.

12. A method for storing data as recited in claim 1 further comprising generating segment information for each of the plurality of data segments, wherein the segment information includes a digital signature.

13. A method for storing data as recited in claim 1 further comprising generating segment information for each of the plurality of data segments; wherein the segment information includes a short identifier that is not likely to be unique.

14. A method for storing data as recited in claim 1 wherein the summary is a Bloom filter.

15. A data storage device comprising:

an input interface adapted to receive a data stream comprising a plurality of data segments; and

a segment redundancy check engine configured to receive a data stream comprising a plurality of data segments, assign an identifier to one of the plurality of data segments, and determine whether one of the plurality of data segments has been stored previously using a summary, wherein the summary is a space efficient, probabilistic summary of segment information.

16. A computer program product for storing data, the computer program product being embodied in a computer readable storage medium and comprising computer instructions for:

receiving a data stream comprising a plurality of data segments;

assigning an identifier to one of the plurality of data segments; and

determining whether one of the plurality of data segments has been stored previously using a summary, wherein the summary is a space efficient, probabilistic summary of segment information.

* * * * *

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
EMC CORPORATION

(b) County of Residence of First Listed Plaintiff _____
(EXCEPT IN U.S. PLAINTIFF CASES)

(c) Attorneys *(Firm Name, Address, and Telephone Number)*
Jack B. Blumenfeld
Morris, Nichols, Arsht & Tunnell LLP
1201 North Market Street; P.O. Box 1347; Wilmington, DE 19899

DEFENDANTS
PURE STORAGE, 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)*

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

REAL PROPERTY
- ❒ 210 Land Condemnation
- ❒ 220 Foreclosure
- ❒ 230 Rent Lease & Ejectment
- ❒ 240 Torts to Land
- ❒ 245 Tort Product Liability
- ❒ 290 All Other Real Property

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

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

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

PROPERTY RIGHTS
- ❒ 820 Copyrights
- ☒ 830 Patent
- ❒ 840 Trademark

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
- ❒ 400 State Reapportionment
- ❒ 410 Antitrust
- ❒ 430 Banks and Banking
- ❒ 450 Commerce
- ❒ 460 Deportation
- ❒ 470 Racketeer Influenced and Corrupt Organizations
- ❒ 480 Consumer Credit
- ❒ 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

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

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. § 271
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 Andrews DOCKET NUMBER 13-1985

DATE 03/21/2016
SIGNATURE OF ATTORNEY OF RECORD /s/ Jack B. Blumenfeld

FOR OFFICE USE ONLY

RECEIPT # _____ AMOUNT _____ APPLYING IFP _____ JUDGE _____ MAG. JUDGE _____

INSTRUCTIONS FOR ATTORNEYS COMPLETING CIVIL COVER SHEET FORM JS 44

Authority For Civil Cover Sheet

The JS 44 civil cover sheet and the information contained herein neither replaces nor supplements the filings and service of pleading 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. Consequently, a civil cover sheet is submitted to the Clerk of Court for each civil complaint filed. The attorney filing a case should complete the form as follows:

I.(a) Plaintiffs-Defendants. Enter names (last, first, middle initial) of plaintiff and defendant. If the plaintiff or defendant is a government agency, use only the full name or standard abbreviations. If the plaintiff or defendant is an official within a government agency, identify first the agency and then the official, giving both name and title.

(b) County of Residence. For each civil case filed, except U.S. plaintiff cases, enter the name of the county where the first listed plaintiff resides at the time of filing. In U.S. plaintiff cases, enter the name of the county in which the first listed defendant resides at the time of filing. (NOTE: In land condemnation cases, the county of residence of the "defendant" is the location of the tract of land involved.)

(c) Attorneys. Enter the firm name, address, telephone number, and attorney of record. If there are several attorneys, list them on an attachment, noting in this section "(see attachment)".

II. Jurisdiction. The basis of jurisdiction is set forth under Rule 8(a), F.R.Cv.P., which requires that jurisdictions be shown in pleadings. Place an "X" in one of the boxes. If there is more than one basis of jurisdiction, precedence is given in the order shown below.
United States plaintiff. (1) Jurisdiction based on 28 U.S.C. 1345 and 1348. Suits by agencies and officers of the United States are included here.
United States defendant. (2) When the plaintiff is suing the United States, its officers or agencies, place an "X" in this box.
Federal question. (3) This refers to suits under 28 U.S.C. 1331, where jurisdiction arises under the Constitution of the United States, an amendment to the Constitution, an act of Congress or a treaty of the United States. In cases where the U.S. is a party, the U.S. plaintiff or defendant code takes precedence, and box 1 or 2 should be marked.
Diversity of citizenship. (4) This refers to suits under 28 U.S.C. 1332, where parties are citizens of different states. When Box 4 is checked, the citizenship of the different parties must be checked. (See Section III below**; NOTE: federal question actions take precedence over diversity cases.**)

III. Residence (citizenship) of Principal Parties. This section of the JS 44 is to be completed if diversity of citizenship was indicated above. Mark this section for each principal party.

IV. Nature of Suit. Place an "X" in the appropriate box. If the nature of suit cannot be determined, be sure the cause of action, in Section VI below, is sufficient to enable the deputy clerk or the statistical clerk(s) in the Administrative Office to determine the nature of suit. If the cause fits more than one nature of suit, select the most definitive.

V. Origin. Place an "X" in one of the six boxes.
Original Proceedings. (1) Cases which originate in the United States district courts.
Removed from State Court. (2) Proceedings initiated in state courts may be removed to the district courts under Title 28 U.S.C., Section 1441. When the petition for removal is granted, check this box.
Remanded from Appellate Court. (3) Check this box for cases remanded to the district court for further action. Use the date of remand as the filing date.
Reinstated or Reopened. (4) Check this box for cases reinstated or reopened in the district court. Use the reopening date as the filing date.
Transferred from Another District. (5) For cases transferred under Title 28 U.S.C. Section 1404(a). Do not use this for within district transfers or multidistrict litigation transfers.
Multidistrict Litigation. (6) Check this box when a multidistrict case is transferred into the district under authority of Title 28 U.S.C. Section 1407. When this box is checked, do not check (5) above.

VI. Cause of Action. Report the civil statute directly related to the cause of action and give a brief description of the cause. **Do not cite jurisdictional statutes unless diversity.** Example: U.S. Civil Statute: 47 USC 553 Brief Description: Unauthorized reception of cable service

VII. Requested in Complaint. Class Action. Place an "X" in this box if you are filing a class action under Rule 23, F.R.Cv.P.
Demand. In this space enter the actual dollar amount being demanded or indicate other demand, such as a preliminary injunction.
Jury Demand. Check the appropriate box to indicate whether or not a jury is being demanded.

VIII. Related Cases. This section of the JS 44 is used to reference related pending cases, if any. If there are related pending cases, insert the docket numbers and the corresponding judge names for such cases.

Date and Attorney Signature. Date and sign the civil cover sheet.