SYSTEM FOR SCALABLE DISTRIBUTED DATA STRUCTURE HAVING SCALABLE AVAILABILITY

Inventors: Witold Litwin, Boulogne (FR); Jaiashankar Mouchedath Menon, San Jose, CA (US); Tore Johan Martin Risch, Linkoping (SE)

Assignee: International Business Machines Corporation, Armonk, NY (US)

Notice: Under 35 U.S.C. 154(b), the term of this patent shall be extended for 0 days.

Appl. No.: 09/083,828
Filed: May 22, 1998

Int. Cl.? ................................. G06F 11/10

U.S. Cl. .............................. 714/7; 714/5; 714/6; 714/7; 714/701; 714/710; 714/711; 714/766; 714/769; 714/770; 711/114

Field of Search ........................... 714/5, 6, 7, 701, 714/710, 711, 766, 769, 770; 711/114

References Cited

U.S. PATENT DOCUMENTS

5,287,362 * 10/1993 Menon ............................... 711/114
5,301,297 4/1994 Menon et al.
5,341,381 8/1994 Fuller
5,506,979 * 4/1996 Menon ............................... 711/112
5,522,032 5/1996 Franaszek et al.
5,542,087 * 7/1996 Neimat ............................. 707/10
5,564,110 10/1996 Aral ................................. 714/7
5,634,033 5/1997 Stewart et al.
5,640,506 6/1997 Duffy
5,875,456 * 2/1999 Stallmo ............................ 711/114
5,875,457 * 2/1999 Shalt .............................. 711/114
5,960,169 * 9/1999 Styczynski ........................ 714/6
6,000,010 * 12/1999 Legg .............................. 711/114

OTHER PUBLICATIONS


“A Case for Redundant Arrays of Inexpensive Disks (RAID)”, by D.A. Patterson, G. Gibson and R.H. Katz, Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley.

“Algorithms for Software and Low-cost Hardware RAID”, by J. Menon, J. Riegel and J. Wylie, IBM Almaden Research Center, IEEE.

(List continued on next page.)

Primary Examiner—Robert W. Beausoleil, Jr.
Assistant Examiner—Bryce Bonzo
(74) Attorney, Agent, or Firm—Altera Law Group, LLC

ABSTRACT

Disclosed is a system for generating parity information for a data file in a distributed data structure system. Data objects in the data file are distributed into data buckets located in memory areas in servers interconnected by a network. An nth set of bucket group numbers are generated. A data bucket and a parity bucket are associated with a bucket group number in the nth set. Parity data for the data objects is generated and stored in a parity bucket associated with a bucket group number in the nth set. After adding a data object to the data file an additional data bucket may be provided for additional data object storage space. After adding a data bucket, a determination is made as to whether bucket availability has decreased below a predetermined threshold. If so, an (n+1)th set of bucket group numbers is generated and parity data for at least one of the data objects is stored in a parity bucket associated with a bucket group number in the (n+1)th set. A bucket group number in the (n+1)th set is associated with a data bucket and a parity bucket.

31 Claims, 9 Drawing Sheets
OTHER PUBLICATIONS


* cited by examiner
**Fig. 6a**

1. **Wait for bucket overflow.**
   - Report overflow to coordinator; coordinator creates new bucket and moves some records to new bucket.

2. **Has bucket exceeded a scaling limit?**
   - **Yes:** Coordinator uses current level of parity files.
   - **No:**
     - Create a new bucket for previous parity files.
     - Create a bucket for first group number in new parity file.
     - Create a new parity file using next grouping function to generate bucket group numbers for new parity file.

3. **For any parity file F(i), does new bucket require bucket group number g(i) not in use?**
   - **Yes:** Create new bucket for new bucket group number g(i).
   - **No:**
Fig. 7a

90 Coordinator notified of a bucket failure.

92 Coordinator creates two lists, L(1) and L(2), including failed data buckets and parity buckets, respectively.

94 Go to first data bucket in list L(1).

96 Coordinator initializes a spare bucket.

98 Set i = 1.

100 Determine bucket group number g(i) for failed bucket B(m) in parity file F(i).

102 Set r = 1.

104 Go to parity record (g, r) in bucket group g in parity file F(i).

106 Does the parity record have only one data key (c)?

110 Coordinator reconstructs data record (c) from parity record and inserts (c) at rank (r) in spare bucket.

112 Are there further records in bucket to recover?

114 Set r = r + 1.

116 Are there further data buckets in list L(1)?

118 Proceed to next failed bucket in bucket list L(1).

120 Go to block 130, in FIG. 6b.

122 Was record reconstructed?

124 Are there further parity files?

126 Set i = i + 1.

128 Recovery fails; more buckets than parity available failed.

Search in data file F(0) for every record c' in record group number (g(i), r) and reconstruct record (c) from located c' and parity data.
Fig. 7b

From FIG. 6a

130. Go to list B(2) of failed parity buckets.

132. Go to first failed parity bucket in list g(i).

134. Determine parity file F(i) and bucket group number g(i) for the failed parity bucket.

136. Create a spare bucket.

138. Set r = 1.

140. Query all data buckets having bucket group number g(i).

142. Request every record R(c) of rank r in each data bucket associated with g(i).

144. Reconstruct parity record (g(i), r) from queried data records R(c).

146. Was recovery of parity record (g(i), r) successful?

148. Are there further records (g(i), r) in parity bucket?

160. Add failed bucket to failed bucket list B(1).

162. Go to block 94 to recover detected failed data buckets.

150. Failed data buckets detected.

152. Set r = r + 1.

154. Are there further failed parity buckets in list B(2)?

156. Go to next failed parity bucket g(i) in list B(2).

158. Recovery of parity and data buckets successfully completed.
SYSTEM FOR SCALABLE DISTRIBUTED DATA STRUCTURE HAVING SCALABLE AVAILABILITY

CROSS-REFERENCE TO RELATED APPLICATION

This application is related to the co-pending and commonly-assigned application Ser. No. 09/083,599, filed on same date herewith, by Witoald Litwin, Jai Menon, and Tore Johan Martin Risch, entitled “Method And System For Data Recovery Using A Distributed And Scalable Data Structure,” which application is incorporated herein by reference in its entirety.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a system for generating parity files in a distributed data structure and, in particular for providing a highly scalable and highly available data structure.

2. Description of the Related Art

The trend in data and file storage is to use distributed data structure systems. In network systems including client and server machines, files may be distributed over many machines in the system. In this way, a pool of computer sites attached to the network provides added power and resources. One recent development is to use the RAM in computers throughout the network system instead of local hard disk drive space. A network system comprised of numerous sites (processors) having megabytes (MB) of RAM per site can provide a distributed RAM space capable of storing files having a size in gigabytes (GB). In network systems, client and server machines function as nodes of a network. Each server (or client) provides storage space within its local RAM or local hard disk drive space to store objects comprising a file. The storage space each machine provides for this network file is called a “bucket.” The number of interconnected servers in a system can extend from 10 to 100,000. The file consists of records, i.e., objects, that are identified by primary keys (e). A record R with a key c denoted as R[c], whereas c refers to the key value.

One problem with such distributed data structure systems is how to determine the optimal number of sites to utilize to store the distributed file. The use of too many sites may deteriorate system performance. Moreover, the optimal number of sites is often unknown in advance or can change as the file size changes. The goals of a distributed data structure include: (1) providing flexibility such that new servers can be added to the distributed system as a file expands; (2) no centralized site that must process and manage all computations; (3) no need to provide updates to multiple nodes in the system to primitive commands, e.g., search, insert, delete, etc. A distributed system that satisfies the above three constraints is known as a Scalable Distributed Data Structure (SDDS).

On such prior art SDDS is the Linear Hashing system, also known as LH*, described in detail in “LH*—A Scalable, Distributed Data Structure,” by Witoald Litwin, Marie-Anne Neimat, and Donovan A. Schneider, published in ACM Transactions on Database Systems, Vol. 21, No. 4, December 1996, pages 480–525, which is incorporated herein by reference in its entirety. An LH* file is stored across multiple buckets comprising local disk drive space or RAM. The LH* is a hashing function that assigns a bucket address to a key c added to the file by applying the hashing function to the key c value.

Each bucket has a predetermined record limit b. When a bucket reaches such predetermined maximum size, a bucket is added to the system and the contents of the full bucket are split into this new bucket. Every split moves about half of the records in a bucket to a new bucket at a new server (site). The bucket to be split is denoted by a pointer referred to as n, the split pointer. Buckets are split sequentially, where the split pointer maintains the last bucket split. The file level q is the file level value that indicates the number of splitting sequences that have occurred. The file level q is used to determine the number of buckets, 2q−1, at any given level. For instance, when there is only 1 bucket, q=0. When a new bucket is created, q increments by one, which in the present case increments to q=1. The pointer n is then set back to bucket 0. Bucket 0 is split, then bucket 1 is split. When bucket number 2q−1 bucket is split, which in this case is bucket 1, there are then four total buckets (0, 1, 2, 3) and q is incremented to two. This process of cycling through the current number of buckets to split buckets is described in “LH*—A Scalable, Distributed Data Structure,” by Witoald Litwin et al., incorporated by reference above.

When a bucket overflows, the client or server maintaining the bucket reports the overflow to a dedicated node called a coordinator. The coordinator applies a bucket load control policy to determine whether a split should occur.

When a record c is added to the file F, a directoryless pair of hashing functions h_n and h_{c+1}, wherein q=0, 1, 2, are applied to the record c to determine the bucket address where the record c will be maintained. The function h_n hashes a key (c) for the data record, which is typically the primary key. The value of the split pointer n is used to determine which hashing function, h_n or h_{n+1}, should be applied to the key c to compute a bucket address for the record c. If the coordinator determines that a split should occur, the coordinator signals a client in the system to perform the split calculations. The client uses the hash functions to hash a key c into a bucket address.

Traditional LH approaches assume that all address computations use correct values for q and n. This assumption cannot be satisfied if multiple clients are used unless a master site updates all clients with the correct values for q and n. LH* algorithms do not require that all clients have a consistent view of q and n. In LH*, each client has its own view of q and n, q’ and n’. These values are only updated after a client performs an operation. Because each client has a different view of q and n, each client could calculate a different address for a record c. In LH*, the client forwards the record c to the server based on the result of the hashing function and the values of q’ and n’ maintained by the client.

The server, where the bucket resides, that receives the record c from the client applies the hashing functions using the values of q’ and n’ at the server to determine if the bucket address for the key c is the address of the target server. If the client sent the key c to the correct bucket, the key c is stored in the bucket. If not, the server calculates the new address and forwards the key c to the correct server (bucket). The recipient of the key checks the address again, and may resend the key c to another bucket if the initial target server (bucket) calculated the wrong address using its values n’ and q’.

Under current LH* algorithms, records are forwarded to a correct new bucket address in at most two forwardings. When the correct server gets the records for the new bucket, the correct server sends an image adjustment message (IAM) to the client, and any intermediary servers using incorrect values of n and q, to allow the client (or server) to adjust its values of n’ and q’ closer to the correct image of q and n.
Such LHS schemes typically guarantee that all data remain available, i.e., recoverable, as long as no more than s sites (buckets) fail. The value of s is a parameter chosen at file creation. Such s availability schemes suffice for static files. However, one problem with such prior art static schemes is that such schemes do not work sufficiently for dynamic files where the size of the file is scalable. For instance, at given value of s, i.e., the system remains available if no more than s buckets fail, as the file increases in size, the file reliability, i.e., probability that no data is lost, decreases.

SUMMARY OF THE INVENTION

To overcome the limitations in the prior art described above, preferred embodiments of the present invention disclose a system for generating parity information for a data file. Data objects in the data file are distributed into data buckets located in storage areas in servers interconnected by a network. An nth set of bucket group numbers is generated. A data bucket and a parity bucket are associated with a bucket group number in the nth set. Parity data for the data objects is generated and stored in a parity bucket associated with a bucket group number in the nth set. After adding a data object to the data file an additional data bucket may be provided for additional data object storage space. After adding a data bucket, a determination is made as to whether bucket availability has decreased below a predetermined threshold. If so, an (n+1)th set of bucket group numbers is generated and parity data for at least one of the data objects is stored in a parity bucket associated with a bucket group number in the (n+1)th set. A bucket group number in the (n+1)th set is associated with a data bucket and a parity bucket.

In further embodiments, prior to generating the (n+1)th set of bucket group numbers, parity data for data objects in k data buckets are maintained in a parity bucket associated with a bucket group number in the nth set. In such case, k is reduced to a lower value k'. Parity data for data objects in k' data buckets is stored in at least one parity bucket associated with a bucket group number in the nth set and parity data for data objects in k-k' data buckets is stored in at least one parity bucket associated with a bucket group number in the (n+1)th set. Data buckets associated with bucket group numbers in the nth set are not associated with bucket group numbers in the (n+1)th set.

In still further embodiments, the parity data generated and stored in parity buckets associated with bucket group numbers in the nth set comprises first parity data. In such case, the step of generating the (n+1)th set of bucket group numbers further includes the step of generating second parity data for the data objects. The step of storing the parity data includes storing the nth parity data in at least one parity bucket associated with bucket group numbers in the nth set and storing the (n+1)th parity data in at least one parity bucket associated with a bucket group number in the (n+1)th set.

In still further embodiments, data buckets are assigned bucket group numbers in the nth and (n+1)th sets of bucket group numbers such that for a first data bucket and a second data bucket, if the first data bucket and second data bucket are associated with the same bucket group number in the nth set of bucket group numbers, then the first data bucket and second data bucket are associated with different bucket group numbers in the (n+1)th set of bucket group numbers. In yet further embodiments, the data objects in a data bucket are assigned a rank value. For a given rank value and bucket group number, a parity record includes parity data for every data object having the given rank value in each data bucket associated with the given bucket group number. The parity records are stored in a parity bucket having the same bucket group number as the given bucket group number of the data bucket.

With the above embodiments, a number of parity buckets are provided to insure that a file of data objects remains highly available even as the size of the file changes. High-availability means that the file remains available even if some of its records or buckets are unavailable.

BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 is a block diagram illustrating a software and hardware environment in which preferred embodiments of the present invention are implemented;

FIG. 2 illustrates the data structure of a data record in accordance with preferred embodiments of the present invention;

FIG. 3 is an illustration of data structure of a data record in accordance with preferred embodiments of the present invention;

FIG. 4 illustrates the data structure of a parity record in accordance with preferred embodiments of the present invention;

FIGS. 5a and 5b are charts showing the reliability P(k) of algorithms in accordance with preferred embodiments of the present invention for a variety of values of p and k and for a file scaling up to M=32K buckets (sites);

FIG. 6 is a flowchart illustrating preferred logic for storing parity data for data objects in a data file classified by a bucket group number in accordance with preferred embodiments of the present invention; and

FIGS. 7a and 7b are flowcharts illustrating preferred logic for recovering failed data and parity buckets in accordance with preferred embodiments of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In the following description, reference is made to the accompanying drawings which form a part hereof, and which is shown, by way of illustration, several embodiments of the present invention. It is understood that other embodiments may be utilized and structural changes may be made without departing from the scope of the present invention.

Hardware and Software Environment

FIG. 1 illustrates a network system 2 comprised of servers 4a, b, c, d. Each server includes a memory 6a, b, c, d. Each memory 6a, b, c, d includes a memory area dedicated to one or more buckets B1, B2, ..., Bm1. Buckets (B) are numbers 0, 1, 2, ..., M-1, where M denotes the current number of buckets. A file is stored throughout the buckets B1 through Bm1. Each server 4a, b, c, d is comprised of a computer, such as a personal computer, workstation, mainframe, etc. Each memory 6a, b, c, d is comprised of a random access memory (RAM) controlled by the computer 6a, b, c, d or a local non-volatile storage area, such as a hard disk drive.

A plurality of client computers 10a, b are capable of accessing records in a file stored across the buckets B0.
through B\textsubscript{M-1}. The client computers 10a, b may be a personal computer, laptop, workstation, mainframe, etc. A coordinator computer 8, including memory 6e, provides file management services for files stored throughout the buckets B\textsubscript{0} through B\textsubscript{M-1}. The client computers 10a, b may search, insert or delete records in the data file. The coordinator computer 8 memory 6e, which may be local RAM or hard disk storage space, may also contain a data bucket B\textsubscript{0} and data records.

A network system 12 provides communication among the servers 4a, b, c, d, the coordinator 8, and the clients 10a, b. The network 12 may be comprised of any suitable network architecture known in the art, such as LAN, Ethernet, WAN, SNA networks, a System Area Network (SAN), such as the ServerNet network offered by Compaq Computer Corporation, Token Ring, LocalTalk, TCP/IP, the Internet, etc. The network may further allow the clients 10a, b or any other connected system to communicate via a remote or wireless connection. In further embodiments, the servers 4a, b, c, d may be linked via a high speed communication SAN and the clients 10a, b may be linked to one or more of the servers 4a, b, c, d and 8 via another network communication system, such as a LAN or Ethernet.

In yet further embodiments, the functions described herein as being performed by the coordinator 8 may be distributed throughout the coordinator 8 and servers 4a, b, c, d if the coordinator 8 and servers 4a, b, c, d include parallel processing software, such as the ORACLE® Parallel Server, described in Oracle publication “ORACLE 7 Parallel Server Concepts and Administration, Release 7.3,” part No. A4252201 (Oracle copyright, 1996), which publication is incorporated herein by reference in its entirety. ORACLE is a registered trademark of Oracle Corporation. Moreover, if the coordinator 8 fails, the functions of the coordinator 8 can be assumed by the surviving servers 4a, b, c,d.

**Bucket Groups**

Each file consists of a plurality of records (R). Each record includes a primary key (c). A primary key is a key that holds a unique value for the record (R). A record (R) with key (c) is denoted as R(c). A data object refers to any whole or part of a data record, including the keys or non-key data. The records of the file are stored in buckets B\textsubscript{0} through B\textsubscript{M-1}, wherein each bucket (B) has a capacity of b records, where b=1. In preferred embodiments, a file maintains one bucket on each server 4a, b, c, d. Each server 4a, b, c, d and client 10a, b, c maintain a file allocation table that provides the network address for each bucket, including the server that holds a particular bucket.

FIG. 2 provides an illustration of the structure of a data record R(c) 20. Every data bucket m has an insert counter (r\textsubscript{m}) 22. This counter r\textsubscript{m} is an integer value representing the order in which the record R(c) was added to the bucket, i.e., the rank of each record R(c) in the bucket. The record counter (r\textsubscript{m}) 20 further includes a primary key (c) 24 portion and non-key data 26. Thus, the first record R(c) has an insert counter (r) 22 of 1, the second R(c) added to the bucket has an insert counter (r) 22 of 2, etc. Each file (F) is a collection of files F\textsubscript{m}, where i=0, 1, 2, 3, ... I. The files F\textsubscript{m} are stored on buckets distributed throughout the network 12. The file F\textsubscript{0} includes the data records R(c). The files F\textsubscript{i} for i=0 are parity files. Each parity file F\textsubscript{i}, for i=1, is comprised of a plurality of buckets. Each file consists of at least F\textsubscript{0}, F\textsubscript{1}. When the coordinator 8 adds buckets to handle additional records R(c) added to a file, parity files F\textsubscript{i} may be added. Included in the coordinator 8 and/or other servers 4a, b, c, d throughout the network 12 is a computer program that implements a family of grouping functions, f\textsubscript{i}, where i=0, 1, 2, that assigns bucket group numbers g\textsubscript{i} to data buckets in F\textsubscript{0}. A bucket group number g\textsubscript{i} is the bucket in parity file F\textsubscript{i} in which parity data is maintained for the data bucket in F\textsubscript{0} assigned the bucket group number g\textsubscript{i}. For instance, if parity file F\textsubscript{1} has two buckets, 0 and 1, and a data bucket in F\textsubscript{0} is assigned the bucket group number g\textsubscript{1}=1, then the parity data for such data bucket is maintained in the second bucket in parity file F\textsubscript{2}.

In preferred embodiments, bucket group numbers are generated in accordance with the following Bucket Grouping Proposition:

bucket group numbers g\textsubscript{i} are assigned to buckets, such that for every bucket address m, m\textsubscript{1}, m\textsubscript{2}, and i, if m\textsubscript{1} and m\textsubscript{2} belong to the same bucket group generated by f\textsubscript{i}, where i=1, ... , then for every f\textsubscript{i}, i\equiv 1, m\textsubscript{1}, and m\textsubscript{2} belong to different bucket groups. A bucket group consists of all buckets sharing a bucket group number g\textsubscript{i} assigned according to the grouping function f\textsubscript{i} that adheres to the Bucket Grouping Proposition. The grouping functions, f\textsubscript{i}, are selected such that every bucket group can have no more than k buckets. When the data file F\textsubscript{0} scales up to a bucket number based on some function of k, the next grouping function g\textsubscript{i+1} is used to assign bucket group numbers g\textsubscript{i+1} to the current set of buckets according to the Bucket Grouping Proposition. The new parity file F\textsubscript{i+1} then stores parity data according to this new assignment of the bucket group numbers g\textsubscript{i} to the current data buckets. For instance, in preferred embodiments, after the kth data bucket is added by splitting, the second grouping function f\textsubscript{2} is used to assign bucket group numbers g\textsubscript{2} to the current set of k buckets. Thus, the parity buckets in parity file F\textsubscript{2} store additional parity data for the data file based on how the grouping function f\textsubscript{2} assigns bucket group numbers g\textsubscript{2} to the data buckets in F\textsubscript{0}. When the data buckets scale up to k\textsuperscript{2}, then grouping function f\textsubscript{3} is used to assign bucket group numbers g\textsubscript{3} to the current number of k\textsuperscript{2} buckets. When the buckets scale up to k\textsuperscript{3} buckets, then parity file F\textsubscript{3} is added and grouping function f\textsubscript{4} is used to assign bucket group numbers g\textsubscript{4} to the current set of k\textsuperscript{3} buckets. Each addition of a parity file F\textsubscript{i} and assignment of bucket group numbers g\textsubscript{i} by an additional grouping function f\textsubscript{i} increases the number of bucket group numbers each data bucket participates in. Thus, if i=2, then each data bucket in F\textsubscript{0} participates in two parity bucket groups and has parity data maintained in two separate parity files. This increases the availability of the data in the bucket in case of failure; otherwise referred to as file availability.

Each value g\textsubscript{i} is calculated according to equations (1) from its data bucket address m and the maximum number of buckets per bucket group k.

\[
g_1 = \text{mod}(m/k)
\]

\[
g_2 = \text{mod}(m/(k^2)) + \text{int}(m/k^2)
\]

\[
g_3 = \text{mod}(m/(k^3)) + \text{int}(m/k^3)
\]

... 

\[
g_i = \text{mod}(m/(k^{i-1})) + \text{int}(m/k^{i-1})
\]

The grouping functions f\textsubscript{i} used to generate bucket group numbers g\textsubscript{i} according to equation (1) are expressed in equations (2):

www.freepatentsonline.com
\[ f_1 : (m, m + 1, m + 2 \ldots m + (k - 1)) \quad \text{for } m = 0, k \ldots \] (2)

\[ f_2 : (m, m + k, m + 2k \ldots m + (k - 1)k) \quad \text{for } m = 0, 1, 2 \ldots k - 1, k^2 \]
\[ k^2 + 1 \ldots k^2 + (k - 1), 2 \times k^2 \ldots \]

\[ f_3 : (m, m + k^2, m + 2k^2 \ldots m + (k - 1)k^2) \quad \text{for } m = 0, 1, \ldots k^2 - 1, k^2, k^2 + 1 \ldots \]
\[ k^2 + (k^2 - 1), 2k^3 \ldots 2k^3 + (k^2 - 1) \ldots \]

\[ f_4 : (m, m + k^{2-1}, m + 2k^{2-1} \ldots m + (k - 2)k^{2-1}) \quad \text{for } m = 0, 1, \ldots k^{2-1} - 1, k^{2-1} \]
\[ k^{2-1} + k^{2-1} - 1, 2k^{2-1} \ldots \]

For example, if \( k=4 \), then the above set of grouping function \( f_i \) equations (2) when \( k=4 \), would assign bucket group numbers \( g_i \) to the left hand addresses of the equals sign and the data bucket addresses assigned to that bucket group number are on the right hand side of the equals sign. Thus, the parity data for the data buckets on the right hand side of the equation is maintained in bucket \( g_i \) of file \( F_i \). For instance, grouping function \( f_1 \) assigns bucket group number 0 to data buckets 0, 4, 8, 12. This means that parity data for data bucket 4 is maintained in bucket 1 of parity file \( F_1 \) and bucket 0 of parity file \( F_2 \).

The bucket group numbers generated by the grouping functions \( f_i \) are on the left hand side of the equals sign and the data bucket addresses assigned to that bucket group number are on the right hand side of the equals sign. Thus, the parity data for the data buckets on the right hand side of the equation is maintained in bucket \( g_i \) of file \( F_i \). For instance, grouping function \( f_1 \) assigns bucket group number 1 to data buckets 4, 5, 6, 7; grouping function \( f_2 \) assigns bucket group number 0 to data buckets 0, 4, 8, 12. This means that parity data for data bucket 4 is maintained in bucket 1 of parity file \( F_1 \) and bucket 0 of parity file \( F_2 \).

FIG. 3 illustrates a two dimensional graph showing how the grouping functions \( f_i \) assign bucket group numbers when \( k=4 \). The points at each \((x, y)\) coordinate are data bucket addresses in \( F_i \). The numbers 0, 1, 2, 3 along the vertical axis are bucket group numbers \( g_i \) assigned by grouping function \( f_i \) to the bucket addresses extending horizontally from each point \((g_i)\) on the vertical axis. Thus, grouping functional \( g_i \) assigns group number \( g_i=0 \) to bucket addresses 0, 4, 8, 12; bucket group number \( g_i=1 \) to bucket addresses 1, 5, 9, 13, etc.

Record Groups

Each record \( R(c) \) in a data bucket is assigned a record group number expressed by the couple \((g_i, r)\). "\( r \)" is the insert number, indicating the rank order in which the record \( R(c) \) is inserted in the data bucket storing the record, and \( i \) is the grouping function number that assigned the group number \( g_i \). In preferred embodiments, a parity group size is at most \( k \). A parity record \((g_i, r)\) is a set of all records \( R(c) \) sharing the same parity record number \((g_i, r)\). A parity record \((g_i, r)\) is stored in the \( g_{i+1} \) bucket of parity file \( F_i \) at insert rank \( r \) within such parity bucket.

FIG. 4 illustrates the file structure of a parity record \((g_i, r)\). The parity record \((g_i, r)\) has a plurality of keys \( c \) from the data records \( R(c) \) in the data buckets. Each parity record \((g_i, r)\) has no more than \( k \) keys \( c \) and parity bits over non-key data to provide error correction in a manner known in the art. The insert counter rank \( r \) may extend from one until the maximum number of records in a bucket \( b \). Thus, each bucket \( g_i \) in the parity file \( F_i \) includes \( r \) records, comprised of parity records \((g_{i-1}, r)\) \ldots \((g_i, b)\) where \( b \) is the maximum number of records in a bucket.

Parity records \((g_i, r)\) need additional storage. There is one parity record \((g_i, r)\) per \( k \) data records \( c \). Thus, a file with \( J \) parity files \( F_i \), for \( i=1 \ldots J \), requires \( Jk \) additional storage buckets. As the number of data buckets in the data file \( F_j \) increases, so does the need for buckets to add parity records \((g_i, r)\).

The preferred embodiment algorithms used to assign bucket and record group numbers to data records and parity files for such records may be implemented as a method, apparatus or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term “article of manufacture” (or alternatively, “computer program product”) as used herein is intended to encompass one or more computer programs and data files accessible from one or more computer-readable devices, carriers, or media, such as a magnetic storage media, “flop disk,” CD-ROM, a holographic unit, a file server providing access to the programs via a network transmission line, etc. Of course, those skilled in the art will recognize many modifications may be made to this configuration without departing from the scope of the present invention.

Measurements of Availability and Alternative Algorithms to Improve Availability

A measure of the availability of the data file as the number of buckets increases is referred to as reliability. Reliability is the probability \( P \) that the entire file \( F_0 \) is available. Higher reliability means lower probability of unavailability, i.e., of an unavailability of \( m+1 \) buckets. Typically, \( P \) is expected to remain above a threshold \( T \), e.g., \( T=0.9 \). Otherwise, a new parity file, i.e., bucket group numbers, will be generated to improve reliability. The probability \( P \) decreases when the number of data buckets in the file \( F_0 \) grows, i.e., when the file scales up. Thus, the reliability of an \( s \) available file decreases as the file scales-up.

In preferred embodiments, a file having \( I \) availability remains available, i.e., failed buckets may be recovered, as long as in every bucket group \( g_i \) there are no more than \( I \) unavaialable buckets. The availability \( P_I \) for an \( I \) available file, wherein \( p \) is the probability that a bucket fails, is:
as much as increasing availability with the orthogonal grouping functions described above. Alternatively, adding a parity bucket, e.g., going to a higher level grouping function using an orthogonal equation, increases the costs because another parity bucket must be added and an additional parity record for each data object must be provided.

In yet further embodiments, a combination of reducing \( k \) and using the orthogonal grouping functions to generate bucket group numbers may be utilized to improve availability. For instance, when availability falls below a predetermined threshold for a first time, \( k \) may be reduced to place fewer data records \( R(c) \) in any given parity bucket group. Subsequently, when availability again falls below a predetermined threshold, the grouping functions \( f_i \) may be used to generate another set of bucket group numbers \( g_i \) to provide that each data record \( R(c) \) now participates in at least \( i \) bucket groups. Any combination of these two algorithms may be used to increase availability after availability falls below the predetermined threshold \( T \).

Generating Parity Records With Grouping Functions

FIG. 6 illustrates logic implemented in hardware or software logic implemented in the coordinator 8 or any other server 4a, b, c, d in the network system 2 for assigning bucket group numbers \( g_i \) to data buckets in the data file \( F_D \); each bucket group number \( g_i \) assigned to the data bucket indicates the bucket number in parity file \( F_P \) that contains parity data for such data bucket. Those skilled in the art will recognize that this logic is provided for illustrative purposes only and that different logic may be used to accomplish the same results.

The logic of FIG. 6 begins at block 40 when an overflow occurs, e.g., a client 10a, b or other device attempt to add a record \( R(c) \) to one of the buckets, \( B_m \) in memory 6a, b, c, d that exceeds the maximum number of records per bucket \( b \). At block 40, the coordinator 8 may already be using \( N \) number of parity files \( F_i \) to \( F_n \) and generating bucket group numbers \( g_i \) through \( g_N \) using grouping functions \( f_i \) through \( f_N \). In such case, control proceeds to block 42 which represents the server 4a, b, c, d reporting the overflow to the coordinator 8 via the network 12. The coordinator 8 then causes a split of one of the buckets \( B_m \) creating bucket \( B_m' \). The coordinator 8 then transfers records from the overflowed bucket to the new bucket \( B_m' \) along with the new record. LH* Methods for splitting buckets upon the occurrence of an overflow are described in “LH*—A Scalable, Distributed Data Structure,” by Witold Litwin et al, which reference was incorporated by reference above.

Control then transfers to block 44 which represents the coordinator 8 determining whether the addition of new bucket \( B_m' \) reaches a scaling limit which indicates that the next grouping function \( f_{N+1} \) function should be used to add another level of availability. For instance, in preferred embodiments, when the number of buckets scales up to \( k' \) for a given \( i \), the next grouping function is applied to generate an additional set of bucket group numbers \( g_{N+1} \). If the new bucket scales \( B_m' \) up to the point where an additional grouping function \( f_{N+1} \) is required, control transfers to block 46; otherwise, control transfers to block 48. In alternative embodiments, other methods could be used to determine whether an additional grouping function \( f_{N+1} \) is required, such as determining whether availability (reliability) has fallen below a predetermined threshold \( T \).

Block 48 represents the state where the coordinator 8 uses the current level of parity files \( F_i \) for \( i=1 \ldots N \), to provide
a parity record for the added data record that caused an overflow. From block 48, control transfers to block 50, which is a decision block representing the coordinator 8, considering the current grouping functions $f_{m,n}$ for $i=1 \ldots N$, determining whether the addition of the new bucket $B_{r,s}$ requires the use of a bucket group number $g_{r,s}$ not currently in use in any of the parity files $F_i$. If so, control transfers to block 52; otherwise control transfers to block 54. Block 52 represents the coordinator 8 creating an additional bucket in the parity file $F_i$ for the new bucket group number $g_{r,s}$ needed for the added data record. This additional bucket will now store parity information for those data buckets in $F_i$ that are assigned this new bucket group number $g_{r,s}$. From block 52 control transfers to block 54, where the process of updating the parity records begins. If a new bucket group number $g_{r,s}$ was not needed to accommodate records from the new bucket $B_{r,s}$, then control proceeds to block 54.

If, at block 44, the coordinator 8 determined that the addition of the new bucket $B_{r,s}$ exceeded a scaling limit, thereby requiring the use of an additional grouping function $f_{m,n}$, then control transfers to block 46. As discussed, an alternative criteria could be used to determine whether to scale-up to the next grouping function $f_{m,n}$, e.g., scaling-up when the availability reaches a threshold minimum. At block 46, the coordinator 8 creates a new parity file $F_{r,s,1}$ and uses the next grouping function $f_{m,n}$ to assign the bucket group numbers $g_{r,s,1}$ to the data buckets. Control then proceeds to block 56 which represents the coordinator 8 creating a new parity bucket to accommodate parity records added to the first bucket, i.e., bucket 0, in the new parity file $F_{r,s,1}$. The coordinator 8 may create a bucket by designating the bucket space in a server 4a, b, c, d to be the next bucket. Control transfers to block 58 which represents the coordinator 8 creating a new bucket in each of the previous parity files $F_i$ for $i=1 \ldots N$. The addition of the bucket triggering the use of the next grouping function $f_{m,n}$ requires each parity file $F_i$ for $i=1 \ldots N$ to add a new bucket group number $g_{r,s}$ for $i=1 \ldots N$ to accommodate the additional bucket $B_{r,s}$ which triggered the scaling upward to another level of availability.

After any data bucket splitting in the $F_i$ file or the addition of parity files $F_i$ or bucket group numbers $g_{r,s}$, control transfers to block 54 et seq. to update the parity records $(g_{r,s})$ maintained in the parity files $F_i$. Block 54 represents the beginning of a loop, wherein the coordinator 8 performs steps 60 through 80 for each parity file $F_i$, for $i \geq 1$. Control proceeds to block 60 which represents the coordinator 8 setting $i=1$ and then to block 62 to set $g_{r,s}=0$. Control proceeds to block 64 which represents the coordinator 8 going to the first parity bucket $g_{r,s}=0$ for a given parity file $F_i$. Control proceeds to block 66 which represents the coordinator 8 determining whether new parity needs to be computed for the parity bucket. If so, control transfers to block 68; otherwise, control transfers to block 67. For instance, the arrangement of records in data buckets prior to the split may not be affected by the split. In such case, the coordinator 8 may use the previously computed parity records for the parity bucket $g_{r,s}$. If parity does not need to be recomputed, then control transfers to block 67 which represents the coordinator 8 assembling the parity bucket $g_{r,s}$ using the previously computed parity records $(g_{r,s})$.

If, however, parity does need to be recomputed, then control transfers to block 68 which represents the coordinator 8 setting the rank $r$ to one ($r=1$). Control transfers to block 70 which represents the coordinator 8 assembling a parity record $(g_{r,s})$ by gathering the key records $c$ from each data record $R(c)$ having rank $r$ in all the data buckets assigned to the bucket group number $g_{r,s}$. The coordinator 8 assigns the gathered key records $c$ to the parity record $(g_{r,s})$. The coordinator 8 further computes parity for the non-key portion of the records $R(c)$ and places the parity data in the parity record $(g_{r,s})$. Control transfers to block 72 which represents the coordinator 8 inserting the assembled parity record $(g_{r,s})$ in the $g_{r,s}$ parity bucket in parity file $F_i$ at rank $r$ in the parity bucket. For instance, if $g_{r,s}=1$, then coordinator 8 would store the parity record $(i, r)$ in bucket 1 of parity file $F_i$ at rank $r$ in the parity bucket 1.

After adding a parity record $(g_{r,s})$ to the parity bucket in parity file $F_i$, control transfers to block 74 which is a decision block representing the coordinator 8 determining whether there are data records in rank $r=1$ in any of the data buckets associated with the current bucket group number $g_{r,s}$. If so, control transfers to block 76; otherwise control transfers to block 78. At block 76, the coordinator 8 increments the current rank $r$ by one and proceeds through steps 70 et seq. to add the next parity record $(g_{r,s}+1)$ to the parity bucket $g_{r,s}$ in parity file $F_i$.

If the previous parity is used at block 67 and if there are no further records $R(c)$ at the next rank $r$ level in the data buckets assigned to the bucket group number $g_{r,s}$ at block 74, then control transfers to block 78 which represents the coordinator 8 determining whether any of the current data buckets in $F_i$ are associated with the next bucket group number $g_{r,s}+1$ for the current parity file $F_i$. If so, control transfers to block 80; otherwise, control transfers to block 82. If there are additional bucket group numbers $g_{r,s}+1$ assigned to data buckets, control transfers to block 80 which represents the coordinator 8 incrementing the bucket group number $g_{r,s}$ by one, i.e., $g_{r,s}=g_{r,s}+1$ and then proceeding to steps 66 et seq. to add all the parity records $(g_{r,s}+1)$ associated with the next bucket group number $g_{r,s}+1$.

If there are no further parity buckets in parity file $F_i$, i.e., no more bucket group numbers $g_{r,s}$ that have not been considered, then control transfers to block 82 which represents the coordinator 8 determining whether there are further parity files $F_i$. If so, control proceeds to block 84 which represents the coordinator 8 incrementing $i$ by one, i.e., $i=i+1$, to update all the parity records $(g_{r,s})$, for the next parity file $F_{i+1}$ at blocks 62 et seq. Otherwise, control transfers to block 84 as updating the arrangement of the parity records $(g_{r,s})$ has been completed.

The logic of FIG. 6 could also be used for file contraction when a data record $R(c)$ is deleted. This operation would be the inverse of the file expansion operations occurring upon a bucket overflow. Deletion of records $c$ could cause the merging of buckets, as described in "LH"—A Scalable, Distributed Data Structure," by Witold Litwin, et al., which was incorporated by reference above. When buckets merge, fewer parity files $F_i$ are needed. When a data bucket in the data file $F_i$ merges, then a related parity bucket in one of the parity files $F_i$ may merge as fewer parity buckets would be needed.

The logic of FIG. 6 and preferred grouping functions $f_{m,n}$ conserve computational resources because the addition of a parity file $F_{r,s}$ to accommodate an added record $R(c)$ does not alter the existing group numbers $g_{r,s}$ assigned to the current data buckets. Only a new group number $g_{r,s}$ is created, or an existing parity record is updated. Increasing the availability of files by adding a parity file $F_i$ does not require a global reorganization of the existing files.

The above logic and algorithms provide for scalable availability, in that the availability, or ability to recover data upon failure, increases as the file size increases. An s-available file means that the file F remains available.
there are further data records R(c) to be recovered, i.e., whether the last rank r was recovered. If so, control transfers to block 114; otherwise, control transfers to block 116. If there are further records R(c) at a higher rank r, control proceeds to block 114 to increment r to recover the next record R(c) at the next rank value r+1 by looping back through steps 104 et seq.

If there are no further records of higher rank r to reconstruct, then at block 116, the coordinator 8 determines whether there are any further failed data buckets in the data bucket list L_r. If there are further failed buckets in the data bucket list L_r, control transfers to block 118 to proceed to the next failed bucket in the bucket list L_r, and return to block 96 to recover the data records R(c) for the next failed data bucket. If, at block 116, the coordinator 8 determines that there are no further data buckets in the list L_r, then control, control transfers to block 120 to proceed to recover any failed parity buckets. Once an entire failed data bucket has been recovered in the spare bucket, the coordinator 8 sends an IAM message to all systems attached to the network 12 to update their physical allocation tables to change the address of the failed bucket to the address of the spare bucket including the reconstructed records.

If, at block 106, the coordinator 8 determined that the parity record (g_r, r) has more than one data record c, control proceeds to block 110 which represents the coordinator 8 searching in data file F_r for every key c in data buckets assigned to the same record group number (g_r, r) to which the failed bucket was assigned that is different from the record c to be constructed. The coordinator 8 then reconstructs record c from the located records c’ in the non-failed data buckets assigned to the bucket group number g_r and parity data in parity record (g_r, r). From block 110, control transfers to block 122 which represents the coordinator 8 determining whether the record c was successfully recovered. If so, control proceeds to block 112 et seq. to recover any further records in the spare bucket, if available; otherwise, control proceeds to block 124.

If the coordinator 8 failed to recover the failed record from the current parity file F_r, then at block 124, the coordinator 8 determines if there are any further parity files F_{r+1} to again attempt to reconstruct the record c. If so, control transfers to block 126; otherwise, control transfers to block 128. At block 126, the coordinator increments i by one to proceed to the next parity file F_{i+1} and back to block 100 et seq. to attempt to reconstruct the record c from the buckets in the next parity file F_{i+1}. In the above logic, the coordinator 8 proceeds sequentially through parity files F_i to recover the failed data by incrementing i at block 126. However, in alternative embodiments, the coordinator 8 could select parity files F_i out of sequence, i.e., in some random or non-sequential order, when attempting to reconstruct the failed record R(c). In such a case, after a parity file F_i is examined and the data record c was not successfully recovered, such considered parity file F_i is not considered again in the recovery of the particular record c. If there are no further parity files and the attempt at recovering the record c failed, control transfers to block 128 which represents the bucket recovery failing. Bucket recovery fails if the number of failed buckets exceeds the availability of parity and data buckets.

The logic of FIG. 7a can be used to recover a key c requested in a key search. If the requested key c is unavailable, then the coordinator 8 can reconstruct key c using the logic of FIG. 7a and then provide that reconstructed key c to the requesting device. In certain embodiments, the coordinator 8 may not restore in the file F_i.
the record R(c) for the recovered key c or any other records R(c) in the failed bucket. Thus, the requested record is recovered only for purposes of the search request.

If all the data buckets in the bucket list L1 have been successfully recovered, control transfers to block 130 in FIG. 7b to reconstruct any failed parity buckets. Block 130 represents the coordinator 8 going to the list L2 containing the failed parity buckets. Control proceeds to block 132 which represents the coordinator 8 going to the first failed parity bucket g. Control transfers to block 134 which represents the coordinator 8 determining the parity file F1 and bucket group number g0 for the failed parity bucket. The coordinator 8 could perform such calculation using the grouping functions. Control then transfers to block 136 which represents the coordinator 8 initializing a spare bucket in which to recover the parity records (g0, r) for the failed parity bucket. Control transfers to block 138 which represents the coordinator 8 setting r, the rank insertion counter, to one. Control then proceeds to block 140 which represents the coordinator 8 querying all data buckets having the bucket group number g0. The coordinator 8 locates the data records R(c) having rank r in the data buckets associated with the bucket group number g0 of the failed parity bucket.

Control transfers to block 142 which represents the coordinator 8 requesting record R(c) having rank r in each data bucket associated with bucket group number g0. Control transfers to block 144 which represents the coordinator 8 reconstructing parity record (g0, r) from the requested records R(c). Control then transfers to block 146 which is a decision block representing the coordinator 8 determining whether recovery of the parity record (g0, r) was successful. If so, control transfers to block 148; otherwise, control transfers to block 150. If the parity record (g0, r) was successfully recovered, then at block 148, the coordinator 8 determines whether there are any additional parity records (g0, r) in the failed parity bucket, i.e., further insert counter values r. If so, control transfers to block 152 to increment the rank r by one; otherwise, control transfers to block 154. From block 152 control proceeds back to block 140 et seq. to recover the subsequent parity record (g0, r+1) in the failed parity bucket associated with bucket group number g0. If, there are no further parity records (g0, r) at block 148, then the recovery of the parity bucket was successful, and control transfers to block 154 which represents the coordinator 8 determining whether there are any further failed parity buckets in the bucket list L2. If so, control transfers to block 156; otherwise, control transfers to block 158. Block 156 represents the coordinator going to the next failed parity bucket g in list L2. From block 156, control transfers back to block 134 to recover the next failed parity bucket in the list L2. If there are no further failed parity buckets in the list L2, then at block 158 all failed parity and/or data buckets have been successfully recovered.

If at block 146 the recovery of the failed parity record was unsuccessful, then the coordinator 8 would have had to detect an unavailable data bucket at block 150. If a parity record or bucket cannot be recovered, then one of the data buckets associated with the failed parity bucket must be unavailable; otherwise, the failed parity bucket would be successfully reconstructed. Control transfers to block 160 which represents the coordinator 8 adding the failed data bucket(s) to the failed data bucket list L1. Control then transfers to block 162 which represents the coordinator 8 returning to block 94 in FIG. 6a to recover the failed data buckets added to the bucket list L1.

In the above algorithm, if the availability is I, i.e., there are I parity files, F1 through F(I), then the algorithm of FIGS. 7a, b almost always terminates successfully. If there are more than I unavailable buckets, then the data recovery operation may fail.

A variation of the above algorithms can be used to recover a single record. For instance, if a client 10a, b initiates a key search and finds that the bucket containing the key c is unavailable, then the client 10a, b would notify the coordinator 8. The coordinator would compute the bucket group number g0 of the failed bucket and send a query to the parity bucket g in parity file F1 requesting the parity record (g0, r) containing c. The coordinator 8 would start at the first parity file, i.e., r=1. If c is the only record in parity record (g0, r), then record c is reconstructed from the parity bits in parity record (g0, r). If there are multiple records c in the parity record (g0, r), then the coordinator 8 searches for all records c' not c in the data buckets in F1 using the hashing function to locate the bucket. If all records c' are located, then the coordinator 8 reconstructs c using all the other records c' and the parity bits.

Initially for data record recovery, the coordinator 8 searches in parity files F1 through F(I). If the above record recovery steps are unsuccessful for F1 through F(I) and there are further parity files F(J) etc. then the next parity file F(J) is examined until the Jth parity file F(J). If there are less than I unavailable buckets, then the coordinator should recover the data record c from one of the existing parity files F(J). If more than I buckets are unavailable, then the record recovery effort will fail.

Conclusion

This concludes the description of the preferred embodiments of the invention.

The following describes some alternative embodiments for accomplishing the present invention.

Preferred embodiments utilize a specific grouping functions. However, in alternative embodiments alternative orthogonal equations that satisfy the Bucket Grouping Proposition could be used. In yet further embodiments, any equation that satisfies the Bucket Grouping Proposition mentioned above could be used.

In preferred embodiments, certain operations are described as being performed by the coordinator 8. However, those skilled in the art will appreciate that other components in the network 12 may be used to carry out some of the grouping and data recovery operations described above as being executed by the coordinator 8.

In summary, preferred embodiments disclose a system for generating parity information for a data file. Data objects in the data file are distributed into data buckets located in memory areas in servers interconnected by a network. An nth set of bucket group numbers are generated. A data bucket and a parity bucket are associated with a bucket group number in the nth set. Parity data for the data objects is generated and stored in a parity bucket associated with a bucket group number in the nth set. After adding a data object to the data file an additional data bucket may be provided for additional data object storage space. After adding a data bucket, a determination is made as to whether bucket availability has decreased below a predetermined threshold. If so, an (n+1)th set of bucket group numbers is generated and parity data for at least one of the data objects is stored in a parity bucket associated with a bucket group number in the (n+1)th set. A bucket group number in the (n+1)th set is associated with a data bucket and a parity bucket.

The foregoing description of the preferred embodiments of the invention has been presented for the purposes of
illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto. The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

What is claimed is:

1. A method for generating parity information for a data file, wherein data objects in the data file are distributed into data buckets located in storage areas in servers interconnected by a network, comprising the steps of:
   - generating an nth set of bucket group numbers, wherein n is an integer value greater than zero, and wherein a data bucket and a parity bucket are associated with a bucket group number in the nth set;
   - generating parity data for the data objects;
   - storing the parity data in a parity bucket associated with a bucket group number in the nth set;
   - adding a data object to the data file;
   - initiating an additional data bucket to provide additional data object storage space;
   - determining whether bucket availability has decreased below a predetermined threshold in response to initiating the additional data bucket;
   - generating an (n+1)th set of bucket group numbers after determining that the initiation of the additional data bucket has caused availability to fall below the predetermined threshold, wherein a data bucket and parity bucket are associated with a bucket group number in the (n+1)th set; and
   - storing parity data for at least one of the data objects in a parity bucket associated with a bucket group number in the (n+1)th set.

2. The method of claim 1, wherein prior to generating the (n+1)th set of bucket group numbers, parity data for data objects in k data buckets are maintained in a parity bucket associated with a bucket group number in the nth set, wherein the step of generating the (n+1)th set of bucket group numbers comprises the steps of:
   - reducing k to a lower value k';
   - associating k' data buckets with a bucket group number in the nth set of bucket group numbers, wherein parity data for the k' data buckets is stored in the bucket group number in the nth set; and
   - generating the (n+1)th set of bucket group numbers, wherein k-k' data buckets not associated with a bucket group number in the nth set are associated with a bucket group number in the (n+1)th set, wherein parity data for the k-k' data buckets not stored in a parity bucket associated with a bucket group number in the nth set is stored in a parity bucket associated with a bucket group number in the (n+1)th set.

3. The method of claim 2, further including the steps of:
   - determining whether bucket availability has decreased below a predetermined threshold for a second time;
   - generating an (n+2)th set of bucket group numbers after determining that bucket availability has decreased below a predetermined threshold for a second time;
   - generating second parity data for the data objects; and
   - storing the second parity data in a parity bucket associated with a bucket group number in the (n+2)th set.

4. The method of claim 1, wherein the parity data generated and stored in parity buckets associated with bucket group numbers in the nth set comprises first parity data, wherein the step of generating the (n+1)th set of bucket group numbers further comprises the step of generating second parity data for the data objects, wherein the step of storing the parity data in a parity bucket associated with a bucket group number in the (n+1)th set comprises storing the second parity data in at least one parity bucket associated with a bucket group number in the (n+1)th set.

5. The method of claim 4, wherein the step of determining whether bucket availability has decreased below a predetermined threshold comprises determining whether the use of an additional data bucket has exceeded a predetermined bucket limit.

6. The method of claim 4, wherein data buckets are assigned bucket group numbers in the nth and (n+1)th sets of bucket group numbers such that for a first data bucket and a second data bucket, if the first data bucket and second data bucket are associated with the same bucket group number in the nth set of bucket group numbers, then the first data bucket and second data bucket are associated with different bucket group numbers in the (n+1)th set of bucket group numbers.

7. The method of claim 4, wherein each data bucket includes at least one data object, wherein each bucket group number is associated with a plurality of data buckets and one parity bucket such that each bucket group number associates a plurality of data buckets with one parity bucket, further comprising the step of:
   - generating first parity data for the data objects in the data buckets associated with the same bucket group number in the nth set of bucket group numbers;
   - storing the generated first parity data in the parity bucket associated with a bucket group number in the nth set;
   - generating second parity data for the data objects in each data bucket associated with the same bucket group number in the (n+1)th set of bucket group numbers; and
   - storing the generated second parity data in the parity bucket associated with a bucket group number in the (n+1)th set.

8. The method of claim 4, wherein each data object in a data bucket is assigned a rank value, further comprising the steps of:
   - for a given rank value and a given bucket group number in the nth set of bucket group numbers, generating a first parity record to include parity data for every data object having the given rank value in the data buckets associated with the given bucket group number in the nth set;
   - storing the first parity record in a parity bucket associated with the given bucket group number in the nth set;
   - for a given rank value and a given bucket group number in the (n+1)th set of bucket group numbers, generating a second parity record to include parity data for every data object having the given rank value in the data buckets associated with the given bucket group number in the (n+1)th set; and
   - storing the second parity record in a parity bucket associated with the given bucket group number in the (n+1)th set.

9. The method of claim 8, wherein data objects are comprised of key data and non-key data, wherein the step of generating a parity record for a given rank value and bucket group number further includes the steps of:
including in the parity record the key data of the data objects whose parity data is maintained in the parity record; and
including in the parity record parity bits used to recover the non-key data of the data objects whose parity data is maintained in the parity record.

10. A computer system for generating parity information for a data file comprised of data objects, comprising:
a plurality of computer devices including storage locations;
a network providing communication among the computer systems;
a plurality of data buckets included within storage locations of the computer devices, wherein the data objects are stored in data buckets throughout the network;
means for generating an nth set of bucket group numbers, wherein n is an integer value greater than zero, wherein a data bucket and a parity bucket are associated with a bucket group number in the nth set;
means for generating parity data for the data objects;
means for storing the parity data in a parity bucket associated with a bucket group number in the nth set;
means for adding a data object to the data file;
means for initiating an additional data bucket to provide additional data object storage space;
means for determining whether bucket availability has decreased below a predetermined threshold;
means generating a second set of bucket group numbers after determining that the initiation of the additional data bucket has caused availability to fall below the predetermined threshold, wherein a data bucket and parity bucket are associated with a bucket group number in the (n+1)th set; and
means for storing parity data for at least one of the data objects in a parity bucket associated with a bucket group number in the (n+1)th set.

11. The computer system of claim 10, wherein the storage location is included in a storage device that is a member of the set of storage devices comprising a non-volatile storage device and a volatile storage device.

12. The computer system of claim 10, wherein prior to generating the (n+1)th set of bucket group numbers, parity data for data objects in k data buckets are maintained in a parity bucket associated with a bucket group number in the nth set, wherein the means for generating the (n+1)th set of bucket group numbers comprises:
means for reducing k to a lower value k';
means for associating k' data buckets with a bucket group number in the nth set of bucket group numbers, wherein parity data for the k' data buckets is stored in the bucket group number in the nth set; and
means for generating the (n+1)th set of bucket group numbers such that k' data buckets not associated with a bucket group number in the nth set are associated with a bucket group number in the (n+1)th set, and parity data for the k-k' data buckets not stored in a parity bucket associated with a bucket group number in the nth set is stored in a parity bucket associated with a bucket group number in the (n+1)th set.

13. The computer system of claim 12, further including:
means for determining whether bucket availability has decreased below a predetermined threshold for a second time;
means for generating an (n+2)th set of bucket group numbers after determining that bucket availability has decreased below a predetermined threshold for a second time;
means for generating second parity data for the data objects; and
means for storing the second parity data in a parity bucket associated with a bucket group number in the (n+2)th set.

14. The computer system of claim 10, wherein the parity data generated and stored in parity buckets associated with bucket group numbers in the nth set comprises first parity data, wherein the means for generating the (n+1)th set of bucket group numbers further generates second parity data for the data objects, wherein the means for storing the parity data in a parity bucket associated with a bucket group number in the (n+1)th set stores the second parity data in at least one parity bucket associated with a bucket group number in the (n+1)th set.

15. The computer system of claim 14, wherein each data bucket includes at least one data object, wherein each bucket group number is associated with a plurality of data buckets and one parity bucket such that each bucket group number associates a plurality of data buckets with one parity bucket, further comprising:
means for generating first parity data for the data objects in the data buckets associated with the same bucket group number in the nth set of bucket group numbers;
means for storing the generated first parity data in the parity bucket associated with a bucket group number in the nth set;
means for generating second parity data for the data objects in each data bucket associated with the same bucket group number in the (n+1)th set of bucket group numbers; and
means for storing the generated second parity data in the parity bucket associated with a bucket group number in the (n+1)th set.

16. The computer system of claim 15, wherein each data object in a data bucket is assigned a rank value, further comprising:
means for generating a first parity record to include parity data for every data object having a given rank value in the data buckets associated with a given bucket group number in the nth set;
means for storing the first parity record in a parity bucket associated with the given bucket group number in the nth set;
means for generating a second parity record to include parity data for data objects having a given rank value in the data buckets associated with a given bucket group number in the (n+1)th set; and
means for storing the second parity record in a parity bucket associated with the given bucket group number in the (n+1)th set.

17. The computer system of claim 16, wherein each data object comprises key data and non-key data, wherein the means for generating a parity record for a given rank value and given bucket group number further includes:
means for generating in the parity record the key data of each data object whose parity data is maintained in the parity record; and
means for generating in the parity record parity bits used to recover the non-key data of the data objects whose parity data is maintained in the parity record.

18. An article of manufacture for use in programming a computer system comprised of a plurality of computer devices interconnected by a network system to generate parity information for a data file, wherein data objects in the
data file are distributed into data buckets located in storage areas in the computer devices, the article of manufacture comprising at least one computer readable storage device including at least one computer program embedded therein that causes computer devices within the computer system to perform the steps of:

- generating an nth set of bucket group numbers, wherein n is an integer value greater than zero, and wherein a data bucket and a parity bucket are associated with a bucket group number in the nth set;
- generating parity data for the data objects;
- storing the parity data in a parity bucket associated with a bucket group number in the nth set;
- adding a data object to the data file;
- initiating an additional data bucket to provide additional data object storage space;
- determining whether bucket availability has decreased below a predetermined threshold in response to initiating the additional data bucket;
- generating an (n+1)th set of bucket group numbers after determining that the initiation of the additional data bucket has caused availability to fall below the predetermined threshold, wherein a data bucket and parity bucket are associated with a bucket group number in the (n+1)th set; and
- storing parity data for at least one of the data objects in a parity bucket associated with a bucket group number in the (n+1)th set.

19. The article of manufacture of claim 18, wherein prior to generating the (n+1)th set of bucket group numbers, parity data for data objects in k data buckets are maintained in a parity bucket associated with a bucket group number in the nth set, wherein the step of generating the (n+1)th set of bucket group numbers comprises the steps of:

- reducing k to a lower value k';
- associating k' data buckets with a bucket group number in the nth set of bucket group numbers, wherein parity data for the k' data buckets is stored in the bucket group number in the nth set; and
- generating the (n+1)th set of bucket group numbers, wherein k'-k data buckets not associated with a bucket group number in the nth set are associated with a bucket group number in the (n+1)th set, wherein parity data for the k'-k data buckets not stored in a parity bucket associated with a bucket group number in the nth set is stored in a parity bucket associated with a bucket group number in the (n+1)th set.

20. The article of manufacture of claim 18, further including the steps of:

- determining whether bucket availability has decreased below a predetermined threshold for a second time;
- generating an (n+2)th set of bucket group numbers after determining that bucket availability has decreased below a predetermined threshold for a second time;
- generating second parity data for the data objects; and
- storing the second parity data in a parity bucket associated with a bucket group number in the (n+2)th set.

21. The article of manufacture of claim 18, wherein the parity data generated and stored in parity buckets associated with bucket group numbers in the nth set comprises first parity data, wherein the step of generating the (n+1)th set of bucket group numbers further comprises the step of generating second parity data for the data objects, wherein the step of storing the parity data in a parity bucket associated with a bucket group number in the (n+1)th set comprises storing the second parity data in at least one parity bucket associated with a bucket group number in the (n+1)th set.

22. The article of manufacture of claim 21, wherein the step of determining whether bucket availability has decreased below a predetermined threshold comprises determining whether the use of an additional data bucket has exceeded a predetermined bucket limit.

23. The article of manufacture of claim 21, wherein data buckets are assigned bucket group numbers in the nth and (n+1)th sets of bucket group numbers such that for a first data bucket and a second data bucket, if the first data bucket and second data bucket are associated with the same bucket group number in the nth set of bucket group numbers, then the first data bucket and second data bucket are associated with different bucket group numbers in the (n+1)th set of bucket group numbers.

24. The article of manufacture of claim 21, wherein each data bucket includes at least one data object, wherein each bucket group number is associated with a plurality of data buckets and one parity bucket such that each bucket group number associates a plurality of data buckets with one parity bucket, further comprising the steps of:

- generating first parity data for the data objects in the data buckets associated with the same bucket group number in the nth set of bucket group numbers;
- storing the generated first parity data in the parity bucket associated with a bucket group number in the nth set;
- generating second parity data for the data objects in each data bucket associated with the same bucket group number in the (n+1)th set of bucket group numbers; and
- storing the generated second parity data in the parity bucket associated with a bucket group number in the (n+1)th set.

25. The article of manufacture of claim 21, wherein each data object in a data bucket is assigned a rank value, further comprising the steps of:

- for a given rank value and a given bucket group number in the nth set of bucket group numbers, generating a first parity record to include parity data for every data object having the given rank value in the data buckets associated with the given bucket group number in the nth set;
- storing the first parity record in a parity bucket associated with the given bucket group number in the nth set;
- for a given rank value and a given bucket group number in the (n+1)th set of bucket group numbers, generating a second parity record to include parity data for every data object having the given rank value in the data buckets associated with the given bucket group number in the (n+1)th set; and
- storing the second parity record in a parity bucket associated with the given bucket group number in the (n+1)th set.

26. The article of manufacture of claim 25, wherein data objects are comprised of key data and non-key data, wherein the step of generating a parity record for a given rank value and bucket group number further includes the steps of:

- including in the parity record the key data of the data objects whose parity data is maintained in the parity record; and
- including in the parity record parity bits used to recover the non-key data of the data objects whose parity data is maintained in the parity record.

27. A plurality of data and parity buckets located in storage areas in computer devices interconnected by a network for access by a computer device within the network, comprising:
a data file structure comprised of a plurality of data
objects stored within data buckets throughout the com-
puter system;
an nth parity file structure including parity data generated
for the data objects in a data bucket, wherein n is an
integer value greater than zero, wherein a bucket group
number in an nth set of bucket group numbers is
associated with a data bucket and a parity bucket,
wherein a parity bucket associated with a bucket group
number in the nth set that is also associated with the
bucket from which the parity data was generated stores
the parity data; and
an (n+1)th parity file structure including parity data
generated for the data objects in a data bucket, wherein
a bucket group number in a (n+1)th set of bucket group
numbers is associated with a data bucket and a parity
bucket, wherein the (n+1)th parity file structure stores
parity data after determining that bucket availability
has decreased below a predetermined threshold.
28. The data and parity buckets of claim 27, wherein prior
to generating the (n+1)th set of bucket group numbers, parity
data for data objects in k data buckets are maintained in a
parity bucket associated with a bucket group number in the
nth set, wherein after generating the (n+1)th set of bucket
group numbers, k data buckets are associated with a bucket
group number in the nth set of bucket group numbers,
wherein k is less than k, wherein parity data for the k data
buckets is stored in the bucket group number in the nth set,
and wherein k-k' data buckets not associated with a bucket
group number in the nth set are associated with a bucket
group number in the (n+1)th set, wherein parity data for the
k-k' data buckets not stored in a parity bucket associated
with a bucket group number in the nth set is stored in a parity
bucket associated with a bucket group number in the (n+1)th
set.
29. The data and parity buckets of claim 27, wherein the
parity data generated and stored in parity buckets associated
with bucket group numbers in the nth set comprises first
parity data, wherein the parity data stored in parity buckets
associated with the (n+1)th set of bucket group numbers
comprises second parity data generated after determining
that bucket availability has decreased below a predetermined
threshold.
30. The data and parity buckets of claim 29, wherein data
buckets are assigned bucket group numbers in the nth and
(n+1)th sets of bucket group numbers such that for a first
data bucket and a second data bucket, if the first data bucket
and second data bucket are associated with the same bucket
group number in the nth set of bucket group numbers, then
the first data bucket and second data bucket are associated
with different bucket group numbers in the (n+1)th set of
bucket group numbers.
31. The data and parity buckets of claim 29, wherein data
objects in a data bucket are assigned a rank value, further
comprising:
parity records, wherein for a given rank value and bucket
group number, each parity record includes parity data
for every data object having the given rank value in the
data buckets associated with the given bucket group
number, wherein the parity records are stored in the
parity bucket associated with the given bucket group
number.