Title: EFFICIENT EXPANSION OF HIGHLY RELIABLE STORAGE ARRAYS AND CLUSTERS

Abstract: Method to flexibly expand a storage array so that the I/O load is uniformly distributed among the new physical storage devices and the amount of copying resulting from the reorganization is minimized. The storage of the array is organized into reliability groups. The data of a reliability group is encoded and stored in blocks distributed among k physical storage devices. The invention supports any type of encoding; including those used by existing RAID levels. The storage array can be expanded by adding one or more physical storage devices. The mapping of reliability groups to physical storage devices is modified to take into account the new physical storage devices using a pseudo random replacement algorithm. The new mapping uniformly distributes the existing storage among the expanded array, yet minimizes the amount of data that must be reorganized.
Cross-Reference to Related Applications

This application is entitled to the benefit and priority of U.S. Provisional Application No. 60/418,792, filed October 16, 2002, the contents of which are incorporated herein by reference.

Field of the Invention

This invention relates to the reliable storage of information and specifically to a method of increasing the capacity of a storage array or cluster, resulting in full utilization of the physical storage devices and minimizing the disruption caused by data reorganization.

Background of the Invention

In the field of information storage, RAID (redundant array of independent disks) and storage clusters are standard techniques for achieving highly reliable storage. RAID systems and storage clusters differ in the type of storage devices aggregated and in the type of interconnect. In RAID systems, disk drives are aggregated and interconnected by a peripheral bus such as ATA or SCSI. In storage clusters, computers (with one or more disk drives) are aggregated and interconnected by a network such as Ethernet. Both techniques organize the physical blocks of storage of a collection of physical storage devices, or storage array, into a linear address space accessible to user programs. User programs access logical blocks of storage in this linear address space in contrast to the physical blocks that reside on the actual storage devices. Logical blocks of the storage array may be accessed directly by user programs, as in many database applications, or may be accessed through a layer of software that provides a file system interface.

The mapping of logical block addresses to physical storage devices and physical block addresses is called the data layout. When combined with redundant storage of information, this mapping allows the system to achieve high resilience from the failure of a hardware component. If a component fails, the mapping can be changed to transparently make use of redundant information maintained in the array.
It is often desirable to support the expansion of a storage array while the system is operational. Procedurally, the physical addition of storage is not a problem. Many RAID systems support hot swappable drives that permit additional drives to be added unobtrusively. And additional storage can be added to storage clusters by configuring the cluster software to recognize an additional computer. The challenges lie in reorganizing the data layout to take advantage of the new physical storage and to do it in a way that results in an organization that maximizes disk throughput while minimizing the overhead in reshuffling data.

The data in a storage array is logically arranged into a sequence of numerically identified data groups. Each data group contains d blocks. To provide resilience to component failure, each data group is reliability encoded and stored as a reliability group consisting of k blocks. There is a one to one correspondence between the data groups that compose the logical view of the array and the reliability groups that are actually stored within the array.

The size of the reliability group is generally larger than the data group to accommodate data replication and/or erasure resilient codes. Each block of the reliability group is stored on one of k logical storage devices so that the loss of individual blocks within a reliability group is statistically independent. Each logical storage device is assigned a specific role in the encoding of the data groups and stores one block of the reliability group. The logical storage devices are implemented by a collection of physical storage devices. The mapping of a logical storage device to a physical storage device can vary with data group number so that concurrent I/O is more uniformly spread across the physical storage devices.

The data layout of a storage array is therefore a mapping of the data group number to a set of physical storage devices and the physical block numbers on those storage devices. The association of a physical storage device with a logical storage device determines the role that the physical storage device will play in the reliability encoding. The physical block address determines where on the physical storage device that the encoded data is stored.

The logical storage devices are often further categorized into data units and check units based on functionality. The data group is striped among the data units
while check units store other redundant copies or erasure resilient codes derived from the user data. If there are d data units and c check units, d blocks of user data are encoded into the k=d+c blocks of the reliability group. While these distinctions are useful when discussing RAID levels, they are specific to a type encoding (systematic codes) – in the general case, all k logical storage devices store erasure resilient codes.

Table 1 Typical mapping of logical to physical storage device mapping for a RAID 5 with 4 storage devices.

<table>
<thead>
<tr>
<th>Reliability Group</th>
<th>L0</th>
<th>L1</th>
<th>L2</th>
<th>L3 (Parity)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 1 is an example of logical to physical storage device mapping from the prior art. Reliability groups are stored on logical drives L0 through L3, where L3 is assigned the role of a check unit. There are 4 physical drives, denoted by 0 through 3, and the assignment of logical drive to physical drive varies with the group number.

Various schemes for encoding the data of a reliability group are known in prior art. (See Chen, P. M., et al., "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys*, 26(2):145-185, 1994, for a survey of RAID techniques, including declustered RAID, the contents of which are herein incorporated by reference). For mirrored (e.g. RAID 1) and other replicated storage systems, all k logical storage devices store a copy of the user data. For RAID 5, the user data is striped among k-1 logical storage devices and the kth logical storage device stores the bit-wise XOR of the data of the other k-1 logical storage devices. For resilience to multiple storage device failures, multiple logical storage devices may be used as check units storing erasure resilient codes.
Regardless of the encoding scheme used, it is always possible to efficiently expand the storage array in increments of \( k \) physical storage devices, as is the common practice. Once the new storage devices are added, they are initialized as a low priority background operation and once initialization is completed, new reliability groups are made available to user programs. The new physical storage devices are used only to add new reliability groups to extend the range of the logical block addresses supported: existing storage does not benefit from the additional I/O throughput provided by the new physical storage devices.

Despite the ability to expand the storage array in this manner, there are disadvantages to this approach. First, it is often not convenient to expand the array in increments of \( k \) physical storage devices, especially when this may require a doubling of the number of physical storage devices. Second, this approach does not make use of the additional I/O performance that could be achieved if the existing data were redistributed over all the physical storage devices.

It is common practice to treat redistribution of the data over additional physical storage devices as an off line maintenance task. The reason is that data layouts commonly are not designed with expansion in mind and the addition of physical storage requires most if not all the data of the array to be rearranged. So it is common practice for the storage array to be taken off line, backed up to tape, reconfigured and finally restored from tape. This practice is time consuming, labor-intensive and exposes the data to risk of data loss.

Nevertheless, the need to distribute the I/O load evenly among all physical devices has become clear in research leading to the development of declustered RAID systems. It is well known that the dedicated parity drive of RAID 4 is a bottleneck for small writes since it must be updated with each I/O to any of the other drives in the reliability group. Consequently, RAID 5 was invented to distribute the parity drive among sets of \( k \) physical drives. But RAID 5 still suffers severe performance degradation when a physical drive has failed. Under these circumstances, a read or write to any storage device in the same reliability group as the failed storage device requires I/O from all the remaining storage devices in the reliability group.
Declustered RAID improves upon RAID 5 by distributing the I/O load uniformly among all the physical storage devices. If there are n physical drives in the system, each of the n takes equal turns in implementing the k logical drives. Declustered RAID systems have optimal distribution of I/O load, but the mapping of declustered RAID systems is more complex, as is the problem of online expansion of such a storage array. For an in-depth discussion of declustered RAID, see M. Holland, et al., "Architecture and Algorithms for On-Line Failure Recovery in Redundant Disk Arrays", Journal of Parallel and Distributed Databases 2, 1994, the contents of which are herein incorporated by reference.

While research in declustered RAID teaches the importance of fully utilizing the I/O capabilities of physical storage devices, it does not teach a method to expand the capacity of a storage array, on line and in a manner minimizing the impact of data reorganization.

**Summary of the Invention**

It is the objective of the present invention to provide a method to flexibly expand a storage array so that the I/O load is uniformly distributed among the new physical storage devices and the amount of copying resulting from the reorganization is minimized. The storage array can be expanded in increments as small as a single physical storage device. No assumptions are made about the encoding of data of a reliability group and so the invention is applicable to mirrored and striped RAID levels and storage clusters.

The invention achieves this objective by a novel application of pseudo random replacement. Given an arbitrary mapping of logical to physical storage devices, a pseudo random number is drawn and used to determine if the mapping is to change and if so which of the logical storage devices is to be remapped to the new physical storage device. The replacement algorithm is repeated for each reliability group and if more than one physical storage device is being added, the replacements are repeated for each new physical storage device.

Once a new map for the reliability groups is determined, the data is reorganized to agree with the new map by copying a minimal amount of data. Only data that has been reassigned to the new physical storage device needs to be moved. Though there are statistical fluctuations, the average amount of data that needs to be
moved is only that needed to rebalance space utilization among the physical storage devices. If the addition of a physical storage device increases the capacity from \( v \) physical storage devices to \( v+1 \) physical storage devices, the average amount of data that needs to be copied for a full array is \( \frac{v}{v+1} \) times the capacity of a single physical storage device.

The reorganization of data can be done without interfering with the operation of the storage array. The copying can be performed as a low priority background task. Until the copying is complete, the old map is used to read data and updates to data go to the storage devices identified by both old and new maps. Once the copy is complete, all reads and updates use the new map. In an alternative embodiment of the invention, a single map is used for both reads and updates, this map switching between that of the old and new maps based on the relation of the reliability group number to the progress of the copy task.

The invention preserves data layout properties important to reliable systems.

If a physical storage device never occurs more than once in any mapping of reliability group to physical devices, then this will be true of the mapping after replacement. If the mapping of logical to physical storage devices is uniform over all the physical storage devices (as in declustered RAID), then this will be true of the mapping after replacement. Both properties of the initial mapping are desirable in a preferred embodiment of the invention.

Nevertheless, the initial mapping of logical to physical storage devices is not specified by the invention nor is the invention tied to a specific implementation. The mapping may be table implemented or algorithmic (i.e. a function of the reliability group number). If the initial mapping is in the form of a table, the invention is applied to the table to create a new table with the additional physical storage devices incorporated into its organization. If the initial mapping is algorithmic, then the pseudo random replacement can be incorporated into this algorithm by seeding the pseudo random number generator with a number derived from the reliability group number. This seeding makes the replacement repeatable when applied to a given reliability group. In a preferred embodiment of the invention, the initial mapping is algorithmic so that the storage overhead associated with tables is avoided.

In its algorithmic form, the performance of the invention is high enough to allow the mapping of logical to physical storage devices to be performed on the fly.
The calculation overhead of computing the mapping for each additional physical storage device is that of computing a uniform deviate from a pseudo random number. Many fast algorithms for computing pseudo random numbers are known, of which the class of algorithms known as linear congruential generators is an example. For a description of pseudo random number generation utilizing linear congruential algorithms, see W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery: "Numerical Recipes in C", 2nd ed., Cambridge University Press, 1992, the contents of which are herein incorporated by reference.

Despite being statistical in nature, the algorithm achieves good storage utilization. The assignment of reliability groups to physical storage devices is random. There is therefore a probability that the storage of one physical storage device will be over committed and require storage allocations to be stopped before all physical devices reach capacity. If a physical storage device can theoretically store data from $N$ reliability groups, the number of reliability groups assigned to the physical storage unit by the invention will be approximately a Poisson distributed random variable with a mean of $N$ and root mean square deviation of $\sqrt{N}$. Typically, $N$ will be of the order of $10^6$ and the under-utilization of the storage array will only be of the order of 0.1%.

**Brief Description of the Drawing**

The invention is described with reference to the several figures of the drawing, in which,

**FIG. 1** is a diagram showing the organization of logical blocks into data groups.

**FIG. 2** is a block diagram showing the encoding of data groups into reliability groups and the dispersal of reliability groups among the logical storage devices.

**FIG. 3** is a diagram showing the organization of data groups into logical segments.

**FIG. 4** is a flowchart for adding an additional physical storage device to the map of logical to physical storage devices.

**FIG. 5** is a diagram showing the addition of a fifth physical storage device to an existing table mapping 3 logical storage devices to 4 physical storage devices.
Detailed Description

The logical/physical storage devices are block oriented and perform I/O in units of blocks. On disk devices, a block is implemented as a fixed number of contiguous disk sectors. The logical block addresses specified by user programs are given in block numbers. The term logical block number is used to differentiate user-specified block addresses from the physical block numbers used by the storage array in specifying I/O to the physical storage devices.

Fig. 1 illustrates the organization of logical blocks into data groups. Each logical block is identified by a logical block number (LBN). Data groups are d blocks in size and consist of d contiguous logical blocks.

Fig. 2 shows the encoding of a data group into a reliability group, which is stored on storage devices. Reliability groups are k blocks in size. The d logical blocks of user data in each data group are encoded into a reliability group and stored on k physical storage devices. Each physical storage device stores one block of the reliability group. The choice of physical storage devices to store a reliability group can vary from one reliability group to another. How the choice of physical storage devices varies with reliability group is part of the data layout of the storage array.

It is convenient to think of the physical storage devices as implementing k logical storage devices. Each logical storage device has the function of storing a specific block of reliability groups. Without loss in generality, the logical storage devices can be identified by the integers [0, k-1]. In this view the encoding of user data onto logical storage devices is constant while the mapping of logical storage devices to physical storage devices varies. The expansion of the storage array is then a problem in changing this device mapping to accommodate additional physical storage devices.

The encoding of data groups into reliability groups is not specified by the invention. In general, this encoding will be simple replication or an erasure resilient code of some type. The choice of encoding affects how individual logical blocks are accessed within a reliability group but is not a consideration in expanding the storage array.
In many embodiments of the invention, the d blocks of each data group will be striped among a portion of the associated reliability group. In this case, logical block number LBN is stored on logical storage device LBN % d where "%" denotes the modulus operator. If the physical storage device that is providing storage for this logical storage device has failed, then it is up to the storage array to reconstruct the lost block using redundant information in the reliability group containing the lost block.

In general, read and write operations on blocks require the storage array to operate on multiple blocks within the same reliability group as the block of interest. The operations necessary on the reliability group are dependent on the encoding method used and are well described in prior art. In the example of RAID 1 (mirroring) encoding, read operations can be satisfied by any of the k duplicate blocks and writes must update all k duplicates.

In addition to the data groups and reliability groups that exist in prior art, the invention organizes the storage array into logical segments. Each logical segment contains Ns data groups (depicted in Fig. 3) that encode into Ns reliability groups. All reliability groups encoding user data from the same logical segment share the same mapping of logical to physical storage devices. The mapping of a logical block number LBN to logical segment number LSN is LSN = LBN / (Ns d) where "/" denotes integer division.

The logical segment number LSN is used to determine the identity of the physical storage devices that contain the data for the reliability group that contains the LBN. If this mapping is table driven, the LSN is just an index into a table consisting of rows of k physical device numbers p0 through pk-1. If this mapping is algorithmic, LSN is the argument to a k-tuple of functions F(x) that return the physical storage device number for logical storage device i and logical segment number x. Without loss in generality, the physical storage device numbers can be taken from the set {0, 1, ..., n-1} where n is the number of physical storage devices. Regardless of whether the implementation is algorithmic or table driven, the LSN specific mapping of k logical storage device numbers to physical storage device numbers is called the device map for the specified LSN.
Many definitions of $F_i(x)$ are known in the literature for mappings that depend on data group number. These can readily be applied to mappings that depend on logical segment number. In a preferred embodiment of the invention, the mapping of logical segment number to physical storage device will be uniform over all $n$ physical storage devices and have substantial variation from one value of LSN to another. An example of such a mapping is $F_i(x) = (x+i) \% n$ where "\%" denotes modulus.

Each physical storage device is also organized into segments. Each physical segment consists of $(N_d/d)$ contiguous blocks. The storage for one logical segment takes one physical segment from each of the physical storage devices. For each physical storage device, a mapping is maintained of the logical segment number to the associated physical segment on the device. The mapping is called the segment map for a specific physical storage device. For a specific physical storage device, the mapping is sparse since only $k$ of the $n$ physical storage devices provides storage for a given logical segment.

In a preferred embodiment of the invention, a memory resident segment map performs the mapping of logical segment numbers to physical segments for each physical storage device. The segment map consists of key-value pairs. The LSN is the key and the physical block number of the start of the physical segment is the value. The data structure implementing the segment map should be apparent to someone skilled in the art. Mathematically, the mapping for physical storage device $i$ can be denoted by the function $G_i(t, x)$ defined such that $G_i(t, LSN) = y$, where $y$ is the physical block number of the start of the physical segment storing data from LSN on physical storage device $i$.

Whenever a block is read or updated, it is generally necessary to know the location of all the blocks that are in the same reliability group as the block of interest. The reliability group will consist of $k$ blocks. Given the logical block number LBN of the block of interest, the block of the reliability group stored on logical storage device $i$ is located at physical block number $PBN_i$ on physical storage device $p_i = F_i(LBN/(N_d/d))$. The $PBN_i$ on $p_i$ is given by $PBN_i = ((LBN/d) \% N_S) + G(p_i, LBN / (N_d/d))$. 
The organization of the storage into segments serves three purposes in the invention. First, by increasing the segment size sufficiently, the number of key-value pairs in the segment maps is reduced so that they may be kept memory resident. Secondly, large segments preserve locality such that blocks that are near one another in logical block addresses tend to be stored in physical blocks that are near one another on the physical storage devices. This increase in locality improves the effectiveness of track caching performed by physical storage devices and I/O buffering performed by the operating system of computers in a storage cluster. Third, larger segments increase the performance of the physical segment copies that take place in order to rebalance the storage array after physical storage devices are added.

The mapping of logical segment numbers to physical segments for each physical device will depend upon the mapping of logical segment numbers to physical storage devices. Once the initial mapping of logical segment numbers to physical storage devices $F(s)$ is established, the mapping of logical segment number to physical segments can be determined by the following:

1. Organize the storage on each physical storage device into physical segments and create a list of free physical segments for each of the physical storage devices $p=[0, n-1]$. All physical storage segments are initially free. The free list stores the physical block number PBN of the start of each free physical segment.

2. For each LSN and for each $i=[0, k-1]$, calculate the physical storage device $p_i = F(s)$, allocate a physical segment on $p_i$ by removing a PBN from the free list on $p_i$ and add the key-value pair (LSN,PBN) to the segment map of physical storage device $p_i$.

Once the segment maps are initialized, they are not changed until additional physical storage devices are added to the storage array. For the purposes of adding physical storage devices, it is convenient to define the following two operations on a segment map. The procedure delete($p$, LSN) removes any key-value pair from the segment map for physical storage device $p$ that may have LSN as its key. The starting PBN of the physical segment associated with the LSN key is put on the list of free physical segments that are on $p$. The procedure insert($p$, LSN) takes a PBN from the list of free physical segments that are on $p$ and adds the key-value pair
(LSN,PBN) to the segment map of p. It is possible that the procedure insert will fail in normal operation due to an over commitment of storage on physical storage device p.

The addition of physical storage devices is done in three stages. First, a new device map is created. Reads and writes to the active storage array continue to use the old device map. Second, for each logical segment the new and old device maps are compared and physical segments are copied to the added physical storage devices so that both old and new device maps retrieve copies of the same data. This copy operation can be performed as a low priority background task. Care needs to be taken that ongoing updates to the data accessed by the old map are reflected in the data accessed by the new map. Third, the storage array switches to the new device map for all read and write access and those physical segments that are only accessible through the old device map are reallocated for use in adding new logical segments to the storage array.

The new device map of LSN to physical storage devices is derived from the old device map using random substitution controlled by a pseudo random number generator used to generate a uniform deviate in the range [0, n-1]. The substitution performed for a particular LSN is independent of that performed for any others. Prior to substituting into the initial map, the pseudo random number generator is seeded by the LSN (or a number derived from the LSN). In subsequent substitutions into a map, the pseudo random number generator is seeded by the last used random number generated for that LSN. This seeding guarantees that the random number sequence is reproducible for a given LSN and will not repeat sooner than the natural period of the generator.

The new device map for a given LSN is created from an existing device map for the same LSN as follows. Given an array M[i] that contains the initial map of logical storage device i=[0, k-1] to a physical storage device in the range [0, n-1], and the sequence of pseudo random numbers R1, R2, ... Rm in the range [0, Q-1], then the addition of m physical drives is accomplished by setting R to each of the numbers R1, R2, ... Rm in turn and performing:

1. Increment the number of physical storage devices n;
2. Set integer \( j \) to \( \lfloor n/R \rfloor /Q \) where the operators \( * \) and / are integer multiplication and division, respectively; and

3. If \( j < k \) then set \( M[j] = n-1 \).

Fig. 4 is a flowchart for this algorithm. The flowchart is applied to the mapping associated with each logical segment. For each logical segment, the flowchart is applied once for each physical storage device to be added. First the pseudo random number generator is seeded \( \textbf{410} \). If the algorithm is being applied to the given LSN for the first time, then the pseudo random number generator is seeded with a value derived from the LSN. Otherwise, the pseudo random number generator is seeded with the last random number drawn for the given LSN. The next random number is drawn \( \textbf{420} \) and used to calculate a uniform deviate \( j \in [0, n-1] \). Here, \( n \) refers to the number of physical storage devices, including the physical device that is in the process of being added. The value of the uniform deviate is compared with \( k \). \( \textbf{440} \). Here, \( k \) refers to the number of logical storage devices. If the uniform deviate \( j \) is less than \( k \), logical device \( j \) is remapped to physical device \( n-1 \). \( \textbf{450} \). Otherwise, the map for the given LSN remains unchanged \( \textbf{460} \).

Fig. 5 illustrates the algorithm being applied to an initial device map \( \textbf{510} \) to produce a new device map that incorporates the additional physical storage devices \( \textbf{520} \). The maps of logical to physical devices are shown as tables, but could be implemented algorithmically. In this example, the column labels L0 through L2 designate the logical storage devices, of which there are three. Individual table entries \( \textbf{530} \) are physical storage device numbers. The LSN \( \textbf{540} \) is the row index into the tables. The initial mapping contains 4 physical storage devices. The algorithm of Fig. 4 is applied to each row in turn. The pseudo random device numbers \( \textbf{550} \) calculated for each row are given. Entries of the table that have changed as a result of the algorithm are circled \( \textbf{560} \). The result is a new device map that maps to 5 physical storage devices.

Once a new map is obtained, the data must be reorganized so that the new device map accesses the same data as the old device map. In one embodiment of the invention, this reorganization can be performed by stepping through all values of logical segment number LSN, comparing \( F_i^{old}(LSN) \) and \( F_i^{new}(LSN) \) for all \( i=[0, k-\ldots] \).
If the values differ for LSN and i, copying the physical segment 
\[ G(F_i^{old}(LSN);LSN) \] on physical storage device \( F_i^{old}(LSN) \) to physical segment 
\[ G(F_i^{new}(LSN);LSN) \] on physical storage device \( F_i^{new}(LSN) \). Except for the device 
and segment maps, which are a novel aspect of the invention, examples exist in prior 
art for doing the copy in a manner such that ongoing updates to the storage array are 
incorporated. The adaptation of these methods to the invention would be apparent to 
someone skilled in the art.

In the final step, the storage array is using the new mapping \( F_i^{new}(x) \) and the 
physical storage segments that are no longer accessible by the user are reallocated for 
use in expanding the range of logical block addresses. The inaccessible physical 
storage segments are first deallocated. This deallocation can be performed by 
stepping through all values of LSN, comparing \( F_i^{old}(LSN) \) and \( F_i^{new}(LSN) \) for all 
\( i=[0, k-1] \). And if the values differ for LSN and i, performing the procedure 
delete(\( F_i^{old}(LSN) \), LSN) on the segment map \( F_i^{old}(LSN) \). Once the deallocations 
have been performed, there will be unallocated physical segments on all the physical 
storage devices that can be used for appending additional logical segments to the 
storage array.

A variety of methods may be used to take advantage of the unallocated 
physical segments. In the preferred embodiment of the invention, whatever method 
that was used originally is used again to extend \( F_i^{old}(x) \) to values of x that were not 
previously achievable because of a lack of physical segments. The original method 
will not take advantage of the added physical storage devices, however, and these will 
need to be added by applying the pseudo random substitution already described. In an 
alternative embodiment, a formula such as \( F_i(x) = (x+i)\%n \) (where \( n \) takes into 
account all the physical storage devices added by the invention) is used directly for all 
logical segment numbers that have been added.

Once this mapping is determined and prior to exposing it to users of the 
storage array, the segment maps will need to be updated. For each new logical block 
number LBN and for each \( i=[0, k-1] \), calculate the physical storage device 
\( p_i = F_i(LSN) \) and perform the procedure insert(\( p_i, LSN \)) on the segment map \( p_i \). The
addition of logical block numbers is complete when the insert procedure fails, indicating there are no more free physical segments on one of the physical storage devices. The storage for the new reliability groups is then initialized, given the reliability encoding scheme in use, and the new storage is made available to users of the storage array.

Other embodiments of the invention will be apparent to those skilled in the art from a consideration of the specification or practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with the true scope and spirit of the invention being indicated by the following claims.

What is claimed is:
A method for expanding the capacity of a storage array which minimizes data reshuffling, the storage of said storage array being provided by n physical storage devices each providing a plurality of physical segments, the storage devices of said storage array organized as k logical storage devices, the storage of said storage array organized as a plurality of logical segments, the storage of each said logical segment being provided by k physical segments located on the k logical storage devices, the mapping of logical storage device to physical storage device for each logical segment being provided by a current device map, said method to expand said storage array of n physical storage devices by adding m additional physical storage devices comprising:

a. initializing the physical segments of each of said m additional physical storage devices to the unallocated state;

b. creating a new device map from the current device map for each said logical segment;

c. said step of creating a new device map from the current device map for said logical segment comprising

representing said new device map by an array M, where array element M[i]∈[0,n-1] specifies the identity of the physical storage device that corresponds to the i-th logical storage device providing storage for said logical segment where i∈[0,k-1],

initializing said new device map to be identical to said current device map,

drawing m pseudo random numbers R₁ through Rₘ in the range [0,Q-1] from a sequence of pseudo random numbers associated with said logical segment,
generating a sequence of integers \( L_j = \left( (n+j) \frac{R_j}{Q} \right) \) for \( j \) in \([1, m]\) 
where \( / \) denotes integer division,

modifying the new device map by testing the values of said sequence 
\( L_j \) and for \( j=1 \) through \( m \) setting \( M[L_j] \) to \( n+j-1 \) if \( S_j < k \),

d. comparing the new device map to the current device map for each said 
logical segment, allocating the physical segments that appear in the 
new device map but not in the current device map and copying the 
physical segments that have been remapped;

e. switching the current device map of said storage array to said new 
device map while retaining the current device map prior to the switch 
as the old device map;

f. de-allocating the physical segments that are accessible through said old 
device map but not through the now current device map; and

g. re-using the storage space provided by said de-allocated physical 
segments to increase the number of logical segments.

2. The method of claim 1 where each logical segment is organized as one or 
more data groups, each data group within the logical segment is associated 
with a reliability group and an \( i \)-th block of the reliability group is stored 
within a physical segment of an \( i \)-th logical storage device for said logical 
segment.

3. A method for organizing storage of a storage array which uniformly 
distributes I/O load and permits incremental expansion without extensive data 
reshuffling, the storage of said storage array being provided by \( n \) physical 
storage devices each providing a plurality of physical blocks each identified 
by a physical block number, the organization of said storage array providing a 
plurality of logical blocks each identified by a logical block number, the 
logical blocks being organized into data groups of \( d \) blocks, each said data 
group being stored as a reliability group consisting of \( k \) blocks, each block of
said reliability group being stored on one of k logical storage devices, said method of organizing said storage array comprising:

a. organizing the logical bocks into logical segments identified by a logical segment number and consisting of \( N_S \) data groups;

b. organizing the physical blocks into physical segments identified by a physical segment number and consisting of \( (N_S \ d) / k \) contiguous physical blocks;

c. for each logical segment number LSN, mapping the k logical storage devices to k distinct physical storage devices; and
d. for each logical segment number LSN, mapping the logical segment number to a physical segment number on each of the physical storage devices associated with the logical segment.

4. The method of claim 3 where the mapping of logical to physical storage devices is based on an algorithm.

5. The method of claim 3 where the mapping of logical to physical storage devices is based on lookup in a table.
Logical Blocks

LBN 0
LBN 1
...
LBN d-1
LBN d
LBN d+1
...
LBN 2d-1

Data Group 0

Data Group 1

FIG. 1
FIG. 2
Logical Blocks

Logical Segment 0

Data Group 0

LBN 0
LBN 1
...
LBN \(d-1\)
LBN \(d\)
LBN \(d+1\)
...
LBN \(2d-1\)
...
LBN \(N_d-1\)
LBN \(N_d\)

Data Group 1

FIG. 3
Seed pseudo random number generator

Calculate next pseudo random number

Calculate random device number j in range [0,n-1]

j < k?

Yes

Re-map logical device j to physical device n-1

No

Keep the map the same

FIG. 4
### FIG. 5

<table>
<thead>
<tr>
<th>LSN</th>
<th>L0</th>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

**Initial Mapping 510**

<table>
<thead>
<tr>
<th>j</th>
<th>L0</th>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**New Mapping 520**

<table>
<thead>
<tr>
<th>j</th>
<th>L0</th>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**530**

**540**

**550**

**560**