Florida International University Technical Report **TR-2006-09-02**

STORM: An Approach to Database Storage Management in Data Center Environments

Kaushik Dutta **Raju Rangaswami**

Florida International University
Miami, FL 33199

kaushik.dutta@fiu.edu raju@cs.fiu.edu

STORM: An Approach to Database Storage Management in Data Center Environments

Kaushik Dutta
College of Business
Florida International University
Miami, FL - 33199, USA
Kaushik.Dutta@fiu.edu

Raju Rangaswami
School of Computing and Information Sciences
Florida International University
Miami, FL - 33199, USA
raju@cs.fiu.edu

Abstract

Database storage management at data centers is a manual, time-consuming, and error-prone task. Such management involves regular movement of database objects across physical storage devices in a Storage Area Network (SAN) so that storage utilization is maximized. We present STORM, an automated approach that guides this task by combining low-overhead information gathering about database access and storage usage patterns, effi cient analysis, and effective decision-making for reconfi guring data layout. The reconfi guration process is guided by the primary optimization objective of minimizing the total data movement required for the reconfi guration, with the secondary constraints of space and balanced I/O bandwidth utilizations across the storage devices. We show that this formulation of the dynamic data layout reconfi guration problem is $\mathcal{NP}$-hard and present a heuristic that provides an approximate solution in $O(Nlog(\frac{N}{M}) + (\frac{N}{M})^2)$ time, where M is the number of storage devices and N is the total number of database objects residing in the storage devices. A simulation study shows that the heuristic converges to an acceptable solution that is successful in balancing storage utilization with an accuracy that lies within 7% of the ideal solution.

1 Introduction

Data center services for medium to large entreprises typically host several petabytes of data on disk drives. Most of this storage houses data residing in hundreds to thousands of databases. This data landscape is both growing as well as dynamic; new data-centric applications are constantly added at data centers, while restrictions such as SOX [14] prevent old and unused data from being deleted. Further, the data access characteristics of these applications change constantly. Ensuring peak application throughput at data centers is incumbent upon addressing this dynamic data management problem in a comprehensive fashion.

Today's IT managers have various storage options ranging from low cost SATA, iSCSI, to high performance RAID storage. Due to the exponential growth in the number of storage devices across data centers and their associated management overhead[1], data center managers are inclining more and more toward isolating storage management at data centers using Storage Area Networks (SAN [11]) - a network whose primary purpose is the transfer of data between computer systems and storage elements. Applications read from and write to storage devices through SAN switches or routers [19].

Although SANs allow significant isolation of storage management from server management, the storage management problem is still complex. Due to the dynamic nature of modern enterprises, the interaction and use of applications changes over time. The dynamic changes in the set of "popular" data results in skewed utilization of network storage devices, both in terms of storage space and I/O bandwidth. Such skewed storage utilization eventually degrades the performance of applications, creating the necessity to buy more storage (when existing storage is not fully utilized), thereby resulting in overall cost increment.

IT managers spend copious amounts of time moving data between storage devices to avoid such skewness. However, manual decision making in large data centers containing several terabytes of data and hundreds of storage devices (if not thousands) is time-consuming, inefficient, and at best sub-optimal. Off-the-shelf relational databases contribute to a large portion of these terabytes of data. Consequently, the manual data management tasks of system administrators mostly involve remapping of database elements (tables, indexes, logs, etc.) to storage devices.

In this paper, we present the architecture and design of STORM, a system that enables automatic identification of skewness in database storage utilization in a data center environment and accordingly proposes an optimal data movement strategy. Moving a large amount of data between

[1]Current estimates put expenditure on storage management at approximately one person per $1 - 10$TB and state that storage cost is dominated by storage management cost rather than hardware cost over the long term [2, 18].

storage devices requires considerable storage bandwidth and time. Though such movement is typically done in periods of low activity, such as night-time, it nevertheless runs the risk of affecting the performance of applications. Moreover, such data movement operations are so critical that they are seldom done in unsupervised mode; a longer time implies greater administrator cost. A longer time requirement for the data movement also prompts data center managers to postpone such activities and live with skewed usage for as long as possible. It is therefore critical to minimize the overall data movement in any reconfiguration operation.

Paper contributions:

1. We present the architecture of STORM, a database storage management system in a data center environment.

2. We present a mathematical model for the problem of of balancing the I/O bandwidth utilization across the storage devices with the objective of minimizing data movement, given the storage device capacity constraints.

3. We show that obtaining an accurate solution to this problem is $\mathcal{NP}$-hard and propose a heuristic algorithm that provides an acceptable approximate solution.

4. We conduct a simulation study to demonstrate the efficiency and accuracy of the heuristic algorithm.

The rest of the paper is organized as follows. We present related research in Section 2. We present a practical data center architecture that incorporates STORM in Section 3. In Section 4, we model the problem of dynamic database storage management in a data center environment. Section 5 presents a heuristic algorithm that provides an approximate solution to this problem, while Section 6 details online techniques for monitoring database and storage usage patterns. Section 7 presents an evaluation of STORM using a simulation study compared against a baseline optimal solution. We make final remarks in Section 8.

2 Related Work

Research on data-placement in parallel database systems [10, 20, 9] may seem related at the first glance. However data placement in a parallel database system is designed with the motivation of achieving maximum query parallelism for a single database system. Whereas our goal is to balance the utilization of shared storage devices in a SAN across multiple database systems, as is typical in a data center setting.

Distributed data storage systems such as Mariposa [22] have developed ways to place data distributed in geographical locations based on access pattern and other cost factors such as network cost. The basic objective of Mariposa and our system are different in that Mariposa works on a single distributed database system, while our system works on multiple centralized database systems, that store data in a shared SAN environment. Further, Mariposa optimizes for a WAN setting where network bandwidth is scarce. In our system database servers are connected to SAN devices over a high-speed network. In such a scenario we can assume all storage devices are equally accessible by database servers. We address the problem of balancing the storage device utilization which is more applicable in a data center environment.

Load balancing server resource usage has been an active area of research for over a decade since early work on web-servers [17]. Traditional load balancers work in a dynamic fashion, operating at a per-request level. Further, they have a single objective i.e. balancing the request load of a set of servers. We address data movement for balancing data-access load with the dual objectives of minimizing the data movement and nullifying the skewness in storage utilization, all while meeting the projected capacity requirement based on future growth of data. Further, such data movement is performed periodically with a much coarser time interval, rather than in a continuous fashion.

Storage management vendors such as Veritas [23], Computer Associates [8], and BMC [5] provide application-independent software for storage management. These solutions typically work at the block level. Any allocation of blocks without application-level knowledge of what the blocks store and how they are being utilization, will likely lead to suboptimal usage of the storage. Further, these solutions involve moving blocks of data from one storage to another to achieve balanced utilization. However, such movement may lead to a single database table being split across several drives, severely complicating the task of a database administrator. As a result, such storage management is seldom used for databases. A similar argument can also be made for research related to data migration [16] across storage devices which migrate data across storages at the block level. Our research focuses on data movement at its application-level granularity such as database tables or indexes, thereby also utilizing the semantic knowledge of the data being moved.

Finally, Oracle's Automatic Storage Manager (ASM) solution [21] proposes a different approach to database storage management by striping each file within a single database across all the available storage using a 1MB stripe size. The claim with ASM is that it eliminates the need to move data dynamically because the striped layout of each file across all drives implicitly balances I/O load. However, this solution works on a per-DB level, requiring a dedicated "disk group" to be allocated to each database. Our solution works in an environment where sharing storage across multiple databases is permitted thereby providing greater utilization of storage resources.

3 System Design and Architecture

We consider a data center environment, with a tiered architecture for providing services, comprising of application servers, database servers, and storage nodes. At the head are the application servers which service user requests. The application servers use the database servers to query the databases, which in turn access data from the tables stored in the SAN device pool. Further, as typically the case, we assume that the datacenter comprises of several database server clusters; these clusters share the storage space provided by the SAN device pool.

Our work focuses on the interaction between the database servers and the SAN devices. Database servers allocate storage space in the SAN devices to store database objects such as tables, indexes, logs, etc. The access patterns for individual objects managed by the database servers are application-dependent and can vary widely over time. Consequently, one of the key problems in managing SAN devices in a database-centric system is to move data from one storage device to another to accommodate the future growth of objects and to balance the utilization of the individual devices in the SAN. This primarily includes reconfiguring the storage allocation and data placement on a per-object basis based on application access patterns. Successful reconfiguration leads to a more balanced access to the SAN device pool by the database servers, thereby preventing bottlenecks at individual storage nodes.

The storage management tasks required for such reconfiguration includes information gathering, analysis, decision making, and execution, akin to the Monitor-Analyze-Plan-Execute loop proposed in the IBM's Autonomic Computing intiative [15]. Currently, each of these tasks are performed manually by system administrators based on human intuition and experience. We propose comprehensive automation of these management tasks in a data center environment. Our system will perform data gathering, analysis, and decision making automatically based on which data-center managers can choose to reconfigure the layout of objects to improve the storage utilization.

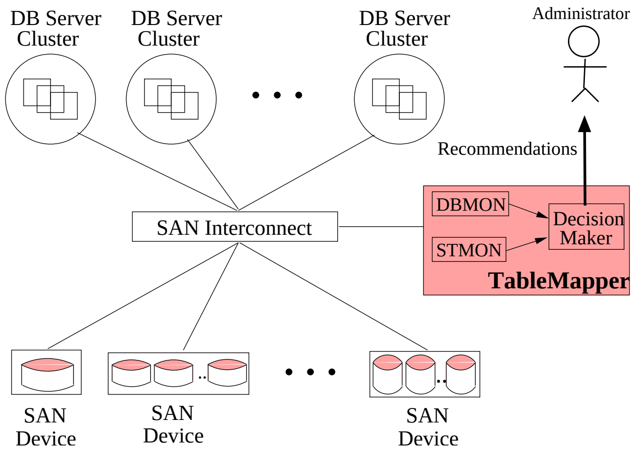

Figure 1. A SAN configuration connecting the database servers and SAN devices. The TableMapper collects usage data (for both the database and storage) periodically for providing reconfiguration hints to the administrator.

The key component of our system is the *TableMapper*, which has access to the database servers as well as the SAN device pool. Figure 1 depicts the TableMapper in a SAN configuration that connects the DB server clusters to the storage devices in the SAN. The TableMapper gathers object-access and storage-usage data (elaborated in Section 6) using the *database monitor* (DBMON) and *storage monitor* (STMON) modules, analyzes the data, and makes reconfiguration decisions within its *decision maker* module. The STMON component gathers data related to storage devices such as storage capacity and I/O bandwidth. These storage data are static in nature and can be adjusted manually when a new storage is added or existing storage is taken out from SAN. The storage utilization, ie which database object is using which storage node and how much space it is consuming, is also gathered by the STMON from database systems. The DBMON component gathers usage information of key database objects (tables and indexes). The data gathering mechanism of both the DBMON and STMON components are described in detail in Section 6. Based on this data, the decision maker analyzes and makes reconfiguration decisions. This analysis and decision-making process are elaborated in Sections 4 and 5. In case a reconfiguration is deemed appropriate by the decision maker, it notifies the system administrator, who may choose to act upon the recommendation.

In realizing the TableMapper we had to consider two key factors. First, the TableMapper must be non-intrusive in collecting object and storage usage information from the database servers. Since this operation is performed periodically and infrequently, it can be performed during system idle time. The second challenge is to avoid a bottleneck at the TableMapper itself. We argue that since the TableMapper only manages metadata and the actual dataflow bypasses the TableMapper, it is unlikely to become a bottleneck. Further, following our architecture, we envision no significant hurdles to using multiple DBMONs to collect data from number of database servers and storage nodes.

The decision maker module of the TableMapper is its most complex component. Based on gathered database-object and storage usage data, the decision maker proposes a reconfiguration of object layout on storage devices. In doing so, it must balance multiple optimization objectives. First, the new configuration should be achievable with minimal data movement. Reducing the total amount of data movement will contribute to realizing the new configuration in lesser time and thus lesser cost, reduce the amount of network traffic, and also reduce the volume of data which may be potentially rendered unavailable during the move. These factors would also encourage data center IT managers to perform more frequent reconfiguration leading to reduced skewness in storage usage over time. Second, it must ensure that none of the storage devices is overly utilized in terms of I/O bandwidth. This is addressed by posing a reconfiguration constraint so that the percentage I/O bandwidth utilization for each device is below the average percentage I/O bandwidth utilization across all storage devices plus a small configurable threshold. Finally, the new configuration should support the future table growth till the storage managers decide for another round of reconfiguration.

In the next section, we formally describe the dynamic storage reconfiguration decision-making problem followed by a heuristic solution (in Section 5) for such decision that will help data center storage manager.

4 Model

We describe the configuration decision making problem formally as *"give a set of database objects (J) with their present growth rate (g_j), usage characteristic (r_j) and size (s_j), and given a set of network storage (I) with allowable I/O bandwidth (U_i^m) and capacity (B_i) specifi cations - determine a new assignment of objects to storage nodes that (i) will result in minimal physical movement of data across storage devices to realize the new assignment, (ii) will balance the I/O bandwidth utilization of storage nodes, and (iii) that will meet the future size growth of objects for certain time (T)."*

Parameter	Description
i	Index for set of physical storage devices (I)
j	Index for set of database objects (J)
c_{ij}	Equals 1, if j is currently located in storage i
s_j	Current size of object j in bytes
g_j	Current growth rate of object j in bytes/days
U_i^m	Maximum I/O bandwidth utilization of the storage in bytes/sec
B_i	Storage capacity in bytes for storage i
r_j	The average bytes/sec retrieved from object j j to serve database requests related to object j
U_{th}	Threshold in percentage that can be allowed for a storage to be over-utilized than the average utilization
T	Validity duration of new object location in days
$\bar{U}$	Average percent utilization of all storages
x_{ij}	Equals 1, if the new allocation of object j is to storage i

Table 1. Model parameters.

Table 1 describes the parameters of the proposed model. Based on these we formulate the dynamic data layout reconfiguration problem, **P**, as follows.

Problem P:

$$\textbf{Z(P)} = min \textstyle\sum_i \sum_j (c_{ij} - x_{ij}) c_{ij} s_j \tag{1}$$

subject to,

$$\textstyle\sum_j (s_j + T g_j) x_{ij} \leq B_i \qquad \forall i \tag{2}$$

$$\bar{U} = 100 \frac{\sum_j \sum_i c_{ij} r_j}{\sum_i U_i^m} \tag{3}$$

$$100 \frac{\sum_j x_{ij} r_j}{U_i^m} \leq \bar{U} + U_{th} \qquad \forall i \tag{4}$$

$$\textstyle\sum_i x_{ij} = \sum_i c_{ij} \qquad \forall j \tag{5}$$

The objective function 1 minimizes the total data movement across storage devices. Constraint 2 ensures that allocated objects have the flexibility to accomodate projected future growth without relocating it to another storage device in future (for T days). Equation 3 computes the average percentage utilization across all storage devices. Constraint 4 ensures that the utilization of each storage node is below the average utilization with a leeway threshold of U_{th}. Constraint 5 ensures that any object that was located in a storage device is assigned to a storage space in the new allocation scheme.

Theorem 1 *The problem* **P** *is $\mathcal{NP}$-Hard*

Proof: We show that problem **P** is $\mathcal{NP}$-Hard by showing that a special case of the problem reduces to the multi-demand constraint multi-dimensional knapsack problem (MDMKP) [7, 6], which is known to be $\mathcal{NP}$-Hard.
The MDMKP problem can be stated in math-programming form as follows

$$max \sum_{j=1,\ldots,n} p_j w_j \tag{6}$$

subject to

$$\sum_{j=1,\ldots,n} a_{ij} w_j \leq b_i, \forall i = 1, \ldots, \forall j = 1, \ldots, n \tag{7}$$

$$\sum_{j=1,\ldots,n} a_{ij} w_j \geq b_i, \forall i = m+1, \ldots, m+q, \forall j = 1, \ldots, n \tag{8}$$

$$w_j \in 0, 1 \tag{9}$$

Let us assume $y_{ij} = x_{ij} - c_{ij}$, then the problem **P** can be written as follows.

$$max \textstyle\sum_i \sum_j y_{ij} c_{ij} s_j$$

subject to

$$\textstyle\sum_j a_j^1 y_{ij} \leq b_i^1 \qquad \forall i$$

where $a_j^1 = s_j + T g_j$ and $b_i^1 = B_i - \sum_j s_j T g_j$ and,

$$\textstyle\sum_j a_j^2 y_{ij} \leq b_i^2 \qquad \forall i$$

where $a_j^2 = r_j$ and $b_j^2 = (\bar{U} + U_{th}) U_i^m / 100 + \sum_j c_{ij} r_j$ and,

$$y_{ij} \leq 0, \qquad y_{ij} \geq 0 \qquad \forall i, j$$

The above form of the problem **P** clearly maps to the MDMKP. Thus we can say the problem can be reduced to a MDMKP in polynomial time. So the problem **P** is also $\mathcal{NP}$-hard. ■

Note that the above problem **P** is an integer programming (IP) problem. Typically IP for large size problems (e.g. thousands of database objects and hundreds of storage devices in a data center) are hard to solve using standard solvers like CPLEX [12]. Further the $\mathcal{NP}$-hardness of the problem **P** makes it harder to obtain exact solutions. In the next section, we develop a simple heuristic algorithm that provides an acceptable approximate solution to the problem with an acceptable time complexity. Moreover unlike CPLEX and the model based approach where we can not get a solution when the problem is infeasible, our heuristic algorithm will provide a solution that will balance the utilization of I/O bandwidth across storage nodes while also meeting the capacity constraint. In Section 7, we evaluate the accuracy of our heuristic.

5 A Heuristic Algorithm

In this section, we present a heuristic algorithm that provides an approximate solution to the problem **P** with an acceptable time complexity. Given the current storage configuration (i.e., the assignment of database objects to individual storage nodes), the heuristic aims at finding a new storage configuration that is better suited to serve the current request load.

The psuedocode for the heuristic algorithm is shown in Figure 2. The algorithm takes as input the current object assignment to storage nodes (c_{ij}), the current bandwidth utilization of each storage node (U_i), and the current I/O bandwidth consumed due to each object (r_i). The algorithm produces as output, a new assignment of objects to storage nodes (x_{ij}). Although this algorithm requires an existing assignment of objects to storage nodes, bootstrapping the system can still be performed by starting with a random assignment of objects to storage nodes.

Greedy heuristics are known to give good heuristic solution for various kinds of knapsack problems [13]. Also a greedy heuristic allows us to develop a simple algorithm that can be easily adapted by data center managers. In our algorithm, we try to move the smaller objects across storage nodes first to achieve the objective goal, before choosing to move the larger ones (i.e. greedy on size). In moving the objects we first try to assign objects with higher bandwidth utilization (r_j) to storage nodes that have lower overall percentage bandwidth utilization ($\frac{U_i}{U_i^m}$), i.e. greedy on I/O bandwidth utilization.

To begin, the algorithm chooses a set S of storage nodes, whose bandwidth utilizations are above the threshold set in Equation 4 or with storage sizes that do not allow the future growth of database objects that they house (line 1). It then creates a list of objects currently residing in the node set S such that removing these objects from S will decrease the utilization of each node in S to an acceptable level. Further, in choosing the objects to place in L, the algorithm ensures that minimum amount of data is moved from the set S of nodes. The creation of L occurs in lines 8 through 17 of the psuedocode. Additionally, for each storage $i \in I$, O_i keeps track of the objects have already been considered for moving into another storage from storage i. This tracking eliminates the reconsideration of the same object, and chooses other ob-

```
Input:      Storage node set ($I$) and database object set ($J$),
            Current configurations ($c_{ij}$, $U_i$, $s_j$, and $r_j$)
Output:     The new assignment of objects to storage nodes ($x_{ij}$)
Constants:  MAX_NUMBER

Begin Algorithm:
1:  Let $S \leftarrow \{\, i \mid i \in I$ and $100\frac{U_i}{U_i^m} > \bar{U} + U_{th}$
        and $\sum_{j \in J} x_{ij}(s_j + Tg_j) < B_i\}$
2:  Let $O_i \leftarrow$ empty, $\forall i \in I$
3:  Let $prevDeviation \leftarrow$ MAX_NUMBER
4:  Let $\bar{U} \leftarrow 100\frac{\sum_j \sum_i c_{ij} r_j}{\sum_i U_i^m}$
5:  Let $currentDeviation = computeDeviation(c_{ij})$
6:  Let $x_{ij} \leftarrow c_{ij} \quad \forall j \in J, i \in I$
7:  While ($S \neq empty$ and $prevDeviation \neq currentDeviation$) {
8:     List $L \leftarrow empty, y_{ij} \leftarrow x_{ij}, \quad \forall i \in I, \forall j \in J$
9:     $\forall i \in S$ {
10:        While ($100\frac{U_i}{U_i^m} > \bar{U} + U_{th}$ or $\sum_{j \in J} x_{ij}(s_j + Tg_j) < B_i$) {
11:          Choose $j \in J$ s.t. $x_{ij} = 1, s_j \notin O_i$, and $s_j \leq s_a$,
               $\forall\, a \in J$ s.t. $x_{ia} = 1$ and $a \notin O_i$
12:          $L \leftarrow L + \{j\}$
13:          $x_{ij} \leftarrow 0$
14:          $O_k \leftarrow O_k + \{n\}$
15:          $U_i \leftarrow U_i - r_j$
16:        }
17:    }
18:    While ($L \neq empty$) {
19:       Choose $n \in L$ s.t. $r_n \geq r_a, \forall\, a \in L$
20:       Let $P \leftarrow \{\, k \mid \sum_j (s_j + Tg_j)x_{kj} + s_n + Tg_n \leq B_m,$
                 $100\frac{U_k + r_n}{U_k^m} \leq (\bar{U} + U_{th})\}$
21:       If ($P$ is empty) {
22:          Choose $k$ s.t. $\frac{U_k + r_n}{U_k^m} \leq \frac{U_l + r_n}{U_l^m}, \forall\, l \in J$ and $n \notin O_k$
23:          If( $k = null$ ) {
24:             $L \leftarrow L - \{n\}$
25:             $x_{in} \leftarrow y_{in} \ \forall i \in I$
26:             goto 18
27:          }
28:          If ($c_{kn} = 1$) {
29:             $O_k \leftarrow O_k + \{n\}$
30:             Choose $o \in I$ s.t. $x_{ko} = 1, o \notin O_k$, and $s_o \leq s_a$,
                   $\forall\, a \in I$ s.t. $x_{ka} = 1$ and $a \notin O_k$
31:             If( $o \neq null$) {
32:                $L \leftarrow L - \{n\} + \{o\}$
33:                $x_{kn} \leftarrow 1$
34:                $x_{ko} \leftarrow 0$
35:                $U_k \leftarrow U_k + r_n - r_o$
36:             }
37:             goto 18
38:          }
39:       }
40:       Else { Choose $k \in P$ s.t. $\frac{U_k}{U_k^m} \leq \frac{U_a}{U_a^m}, \forall\, a \in P$ }
41:       $L \leftarrow L - \{n\}$
42:       $x_{kn} \leftarrow 1$
43:       $U_k \leftarrow U_k + r_n$
44:       $O_k \leftarrow O_k + \{n\}$
45:    }
46:    $S \leftarrow \{\, i \mid i \in I$ and $100\frac{U_i}{U_i^m} > \bar{U} + U_{th}$
        and $\sum_{j \in J} x_{ij}(s_j + Tg_j) < B_i\}$
47:    $prevDeviation \leftarrow currentDeviation$
48:    $currentDeviation \leftarrow computeDeviation(x_{ij})$
49: }
End Algorithm
```

Figure 2. Dynamic data movement heuristic for load-balancing storage node accesses.

```
Input:   Current configurations $x_{ij}$
Output: $deviation$

Begin Algorithm:
1: Let $\bar{U} \leftarrow 100\frac{\sum_j \sum_i c_{ij} r_j}{\sum_i U_i^m}$
2: Let $deviation \leftarrow 0$
3: $\forall i \in I$ {
4:    If( $\bar{U} + U_{th} < 100\frac{\sum_j x_{ij} r_j}{U_i^m}$) {
5:        $deviation \leftarrow deviation + \sum_j x_{ij} r_j - \frac{(\bar{U}+U_{th})U_i^m}{100}$
6:    }
7: }
End Algorithm
```

Figure 3. Computing the deviation between successive solutions of the heuristic.

jects if the attempt to move a smaller object did not succeed in the previous iteration.

In the next phase (lines 18 through 45), the algorithm places each object in list L in a storage node such that the the variance in the bandwidth utilizations is reduced, while considering node capacity with object size and growth requirements as specified in Equation 2. To do so, it maintains the objects in L sorted by their bandwidth utilizations, choosing first the object (n) with the highest utilization (line 19). It then creates a target set P of nodes that can house object n, given the node utilization threshold constraint according to Equation 4 (line 20). Lines 21 through 39 address the case when P is an empty set (described below). Otherwise a node with least percentage I/O bandwidth utilization in P is chosen to house object n (line 40) and requisite book-keeping is performed (lines 41 through 44). After all elements in L have been placed, S and the *deviation* (explained later on) of the I/O bandwidth utilization of storage nodes are recomputed. If S is found to be non-empty and the deviation of bandwidth utilization is different than in the previous iteration, this process is repeated. Otherwise the algorithm terminates with the new assignment given by x_{ij}.

To handle the case when P is empty, we choose a storage node k such that (i) k can accomodate object n given the size constraint, and (ii) placing object n in storage node k causes the least increase in percentage bandwidth utilization across all the storage nodes where the object n was not considered before for storage node k (line 22). If no such storage node k is found, we leave the object n to where it was initially assigned at the beginning of the iteration and go back to the beginning of the while loop to start iterating for the other objects in L. If node k currently houses object n ($c_{kn} = 1$), the algorithm removes n from list L and chooses a different object (o), which has the least size of the remaining objects in k, to place in L, and performs requisite book-keeping (lines 28 through 38). To avoid reconsideration of n in the future it puts the object n into the list O_k of storage node k.

Computing deviation. The algorithm presented in Figure 3 computes the total deviation of I/O bandwidth utilization across all storage nodes for which the percentage I/O bandwidth utilization exceeds the average percentage utilization by more than the utilization threshold U_{th}. We use this value to observe how the over-utilization of storage bandwidth decreases in each iteration of the algorithm.

A key feature of the proposed algorithm is that it obtains a solution even in case it is infeasible to achieve an allocation scheme based on the model. In case the algorithm does not find a feasible solution wherein the percentage I/O bandwidth utilization of all storage nodes are below that threshold value, the algorithm terminates when the deviations in storage utilization in two consecutive iterations remain unchanged. Thereby, in cases when the percentage utilization of I/O bandwidth for each storage node cannot be reduced to satisfy the utilization bound (Equation 3) and meet the capacity constraint (Equation 2), the heuristic still provides a solution that reduces the over-utilization of I/O bandwidth (i.e. deviation) across the storage nodes.

Heuristic complexity. Due to space constraints, we provide only a synopsis of the complexity calculation here. The outer loop (line 7) is repeated at most C times, a positive integer value that depends on the convergence rate of the algorithm. The outer loop at line 9 will be executed $O(|I|)$ time. The loop at line 10 is executed on average $\frac{|J|}{|I|}$ times (average number of objects per storages). Line 11 can be executed in $O(log(\frac{|J|}{|I|}))$. Thus the average order of complexity for the loop at line 10 for each iteration of loop at line 7 is $O(|I|\frac{|J|}{|I|}log(\frac{|J|}{|I|}))$. The inner loop (line 18) is repeated on average $\frac{|J|}{|I|}$ times. Further for each iteration of the inner loop (line 18), line 19 has the average time complexity of $O(\frac{J}{I})$. Line 20 has the complexity of $O(log(|I|))$. One of line 22 or line 40 is executed in each iteration of the inner loop (line 18), both of which have complexity of $O(|I|)$. Line 30 is executed in small percentage of cases where $c_{kn} = 1$. Let us assume it is executed for δ fraction of total execution of the outer loop (line 18). Line 30 has a complexity of $O(log(\frac{|J|}{|I|}))$. Line 46, which belongs to the outer loop has time complexity of $O(|I|)$. The total average time complexity of the algorithm is thus computed to be $O(C \times (|I|\frac{|J|}{|I|}log(\frac{|J|}{|I|}) + \frac{|J|}{|I|}(\frac{|J|}{|I|} + |I| + \delta log(\frac{|J|}{|I|}) + |I|))$. Considering that C is a small number (typically around 5-6, based on experimental data), and $|J| > |I|$ (i.e number of database objects is larger than number of storage nodes, a typical case) we conclude the total time complexity of the heuristic algorithm is $O(|J|log(\frac{|J|}{|I|} + (\frac{|J|}{|I|})^2)$. Thus, our heuristic algorithm is low order polynomial and can be run quickly for large number of database objects and storage nodes.

6 Monitoring Usage Patterns

In this section we describe how STORM collects usage statistics of database objects and their storage consumptions from commercially available databases. These statistics capture the usage behavior of standard classes of database objects (tables and indexes) found in DBMSs. Note that these usage patterns are very often the objects of enquiry for DBAs becuase they lend significant insight into how database objects should be placed in various storage devices across the network. Specifically, STORM collects three *base usage statistics* automatically, i.e.,

1. Frequency of access of database objects.

2. Average data access size of queries for a database object.

3. Storage consumption and growth estimate of objects.

In the STORM architecture, as presented in Figure 1, the DBMON and STMON are profiling modules for the database and storage system respectively. Both processes are configured with a list of database servers (encompassing all the database server clusters) to be monitored. For monitoring database access patterns, the DBMON submits a set of monitoring queries to the database servers, and receives the results. The DBMON then processes these results to yield the monitoring information of interest. The STMON also queries the database servers and the storage devices to obtain dynamic information about space usage for individual database objects by each database server and static information such as storage capacity of individual devices .

The above modules are designed so that they are non-intrusive. The data queried in the monitoring process is not application data, but rather system meta-data. As a result, not only is the total volume of data collected small but also there is minimal contention with application data access. Further, the query interval is typically large and also configurable; it can be made sensitive to database resource availability (e.g., can be done during off-peak hours or service down-time). Finally, once the query results have been received, all other processing takes place outside the database process.

Next, we describe specific practical techniques to gather the above information that apply to most commercial off-the shelf (COTS) databases. We specifically describe our approach for Oracle and SQL-Server databases, but note that our scheme can easily be extended to other COTS databases such as MySQL, DB2, and Sybase.

6.1 Details of Data Gathering Technology

We now describe a specific set of SQL queries that can be used by DBMON and STMON to derive usage statistics of database objects and their storage consumption. This approach is applicable for all COTS databases; however, the specific SQL queries needed to gather are dependent on the meta-data architecture of each vendor's DBMS.

6.1.1 Database Object Access Patterns

Table 2 lists specific queries that DBMON uses to obtain the frequency of accesses of database *table* and *index* objects. From the output of Query Q1, DBMON can generate a list of tables (e.g., CUSTOMER) and columns prefaced with their associated table names (e.g., CUSTOMER.ZIPCODE) in the database. We call these two lists the *TableList* and *ColumnList*, respectively. From the output of Query Q2,

	DBMS	Query
Q1	Oracle	SELECT TABLE_NAME COLUMN_NAME FROM DBA_ALL_TABLES
Q1	SQL-Server	SELECT DATABASE_NAME.DBO.SYSOBJECTS.NAME TABLE_NAME, DATABASE_NAME.DBO.SYSCOLUMNS.NAME, COLUMN_NAME FROM DATABASE_NAME.DBO.SYSOBJECTS, DATABASE_NAME.DBO.SYSCOLUMNS WHERE DATABASE_NAME.DBO.SYSOBJECTS.ID=DATABASE_NAME.DBO.SYSCOLUMNS.ID AND DATABASE_NAME.DBO.SYSOBJECTS.XTYPE = 'U'
Q2	Oracle	SELECT c.INDEX_NAME, c.COLUMN_NAME, c.TABLE_NAME i.NUM_ROWS FROM DBA_IND_COLUMNS c, DBA_INDEXES i WHERE c.INDEX_NAME=i.INDEX_NAME
Q2	SQL Server	SELECT I.NAME INDEX_NAME, I.INDID, O.NAME TABLE_NAME FROM DATABASE_NAME.DBO.SYSINDEXES I, DATABASE_NAME.DBO.SYSOBJECTS O WHERE I.ID = O.ID AND INDID $>$ 0 AND INDID $<$ 255 AND O.TYPE = 'U' ORDER BY O.NAME INDEX_COL (TABLE_NAME , I.INDID , key_id) /* returns the column name where key_id is the ID of the key*/
Q3	Oracle	SELECT DISTINCT s.SQL_TEXT SQL_TEXT, s.EXECUTIONS EXECUTION_COUNT FROM V$SQL s
Q3	SQL Server	SELECT SQL SQL_TEXT, USECOUNTS EXECUTION_COUNT FROM MASTER.DBO.SYSCACHEOBJECTS, MASTER.DBO.SYSDATABASES WHERE UID $>$1 AND ((CACHEOBJTYPE='EXECUTABLE PLAN' AND OBJTYPE='PREPARED') OR (CACHEOBJTYPE='EXECUTABLE PLAN' AND OBJTYPE='PROC' AND SQL NOT LIKE 'SP_%')) AND SYSCACHEOBJECTS.DBID = SYSDATABASES.DBID AND MASTER.DBO.SYSCACHEOBJECTS.DBID=DB_ID('DATABASE_NAME')

Table 2. Database object-usage monitoring queries.

DBMON generates a list of indexes, where each index is identified by the table and column it indexes, (e.g., CUSTOMER.ZIPCODE_INDEX). We call this list the *IndexList*.

Q3 returns a list of SQL statements executed on the database, and each statement's execution count. We call this list the *ExecutedStatementList*. Specifically, each item in this list represents a single statement, and contains the StatmementText (e.g., Select distinct ZIPCODE from CUSTOMER) and an integer StatmentExecCount. For Q3, we note that some consideration must be given to the query submission interval. The tables in Oracle and SQL Server that store executed statements and execution counts are part of the DBMS's caching infrastructure, and are flushed at an administrator-determined interval. The interval between issuing Q3 queries must be smaller than this interval.

DBMON can determine database object usage statistics by *aggregating access counts*. The pseudocode in Algorithm 1 describes this aggregation process.

Lines 1-6 create a set of aggregation structures: the set of $count_t$ variables store the number of accesses to each table, and the sets of column variables $countIndexUsed_c$ and $countIndexNeeded_c$ represent the number of accesses to columns for restriction purposes, where an index would be used if available. The *restrictionUseHashTable* structure stores restriction uses for specific column values and $RetDataSize_t$ represents the total amount of data returned from a table on a set of queries. Lines 8-9 parse the query into tables and restriction columns and column-values accessed, while lines 12-23 update the $count_t$, $countIndexUsed_c$, $countIndexNeeded_c$, and *restrictionUseHashTable* with the execution counts for each query. With this information, DBMON is now prepared to compute the usage statistics that will be required by our approach in Section 5 for the *decision maker* of STORM.

The frequency of access of database objects. The $count_t$ and $countIndexUsed_c$ value give us the usage statistics for database objects tables and indexes.

Algorithm 1 Aggregate usage statistics

```
for all tables $t$ in $TableList$ do
    Create a $count_t$ variable, and initialize it to 0
    Create a $RetDataSize_t$ variable and initialize it to 0
for all columns $c$ in $ColumnList$ do
    Create a $countIndexUse_c$ variable, and initialize it to 0
    Create a $countIndexNeeded_c$ variable, and initialize it to 0
Create an empty $restrictionUseHashTable$
for all statement $s$ in $ExecutedStatementList$ do
    Create three empty lists, $tableAccessList$, $columnAccessList$, and $columnValueList$
    Parse $s$'s StatementText. Add all table names accessed in the WHERE clause to the $tableAccessList$, and all column names accessed in the WHERE clause to the $columnAccessList$. Add column name-value pairs for all column values accessed in the WHERE clause to the $columnValueList$
    Based on the range values and selectivity analysis ([4, 3, 1] DBMON estimates the size of return data set for each table in $tableAccessList$
    Update $RetDataSize_t$ based on computed return data size and StatementExecCount
    for all tables in the $tableAccessList$ do
        Increment $count_t$ by $s$'s StatementExecCount
    for all columns in the $columnList$ do
        if the column name matches an index in the $IndexList$ then
            Increment $countIndexUsed_c$ by $s$'s StatementExecCount
        else
            Increment $countIndexNeeded_c$ by $s$'s StatementExecCount
    for all items in the $columnValueList$ do
        if the column-value pair does not exist in the $restrictionUseHashTable$ then
            Create a new hash element, with the column-value pair as the key, and $s$'s StatementExecCount as the value
        else
            Increment the hash element value for the column-value pair by $s$'s StatementExecCount
Return the $restrictionUseHashTable$, $count_t$ for all tables, as well as $restrictionUse_c$, $countIndexUsed_c$ and $countIndexNeeded_c$ for all columns
```

Average data return size of queries for a database object. Combining the *count*$_t$ along with *RetDataSize*$_t$ DBMON can accurately compute r_j, the average bytes/sec retrieved from the object (table).

Cache considerations. Modern database systems cache frequently accessed objects in memory. This introduces an error into our original calculations where we assume that all data retrieved for query computations are from storage devices. The effect of cache accesses can be taken into account in the exact computation of the usage statistics in the following way. Table 3 shows the query and the system stored procedures in Oracle and SQL-servers respectively that provide specific statistics about cache hits. The oracle query gives list of queries that required disk access along with number of such disk access. The SQL-Server stored procedure gives information about the queries that are being served from the buffer cache, i.e. disk access is not required. These can be used by DBMON along with previously obtained information to accurately compute the average bytes/sec retrieved (r_j) from various database objects, taking into account the effect of the database buffer cache.

DBMS	Query
Oracle	SELECT executions, buffer_gets, disk_reads, first_load_time, sql_text FROM v$sqlarea ORDER BY disk_reads
SQL-Server	DBCC MEMUSAGE

Table 3. Cache usage queries.

6.1.2 Storage Consumption Patterns

	DBMS	Query Description	Query
Q4a	Oracle	Tablespace consumption	SELECT df.TABLESPACE_NAME, SUM(df.BYTES) TOTAL_SPACE, SUM(fs.BYTES) FREE_SPACE, ROUND(((NVL(SUM(fs.BYTES),0)/SUM(df.BYTES))*100),2) PCT_FREE FROM DBA_FREE_SPACE fs, DBA_DATA_FILES df WHERE df.TABLESPACE_NAME = fs.TABLESPACE_NAME (+) GROUP BY df.TABLESPACE_NAME ORDER BY df.TABLESPACE_NAME
Q4b	Oracle	Physical files associated with a Tablespace	SELECT FILE_NAME, TABLESPACE_NAME FROM DBA_DATA_FILES WHERE STATUS='AVAILABLE';
Q4	SQL Server	Physical files associated with a File-group	SELECT FILEGROUP_NAME(GROUPID), GROUP_NAME, FILESIZE TOTAL_SPACE, FILEMAXSIZE-FILESIZE FREE_SPACE, FILEMAXSIZE, GROWTH = (CASE SYSFILES.STATUS & 0X100000 WHEN 0X100000 THEN CONVERT(NVARCHAR(3), GROWTH) + N'%' ELSE CONVERT(NVARCHAR(15), GROWTH * 8) + N' KB' END), NAME LOGICAL_FILENAME ,FILENAME FROM DATABASENAME.DBO.SYSFILES WHERE GROUPID $<>$ 0
Q5	Oracle	Size of index and tables	SELECT sum(bytes)/1048576 Megs, segment_name FROM user_extents WHERE segment_name = 'object_name' GROUP BY segment_name
Q5	SQL Server	Size of index and tables	EXEC sp_spaceused 'table_name' @updateusage = 'true';

Table 4. Storage consumption queries.

Table 4 lists queries used by STMON to obtain information about object storage consumption and storage consumption growth (base usage pattern **3**). For Oracle, STMON generates a list of tablespaces, where each tablespace is associated with the total space for the tablespace, free space, and the percentage of free space from the output of query Q4a in Table 4. We can add the filename(s) associated with the tablespace from the output of query Q4b. For SQL Server, all these data elements can be obtained for filegroups from Q4 for SQL Server. Note that the expression in parenthesis (CASE...END) in this query is an embedded stored procedure designed to convert binary-format numbers into integer format. With this information STMON can compute c_{ij} (current assignment of database object to storage nodes).

The query Q5 gives us the size of database objects tables and indexes (s_j) in Oracle and SQL-Server. Taking a series of these values over time provides data points which STMON uses to forecast growth of table size over time (g_j).

7 Simulation Results

In this section, we experimentally demonstrate the efficacy of our heuristic algorithm in terms of two key metrics (i) accuracy, and (ii) convergence.

7.1 Accuracy of Heuristic Algorithm

To evaluate the accuracy of the heuristic, we compare the data movement required by the heuristic solution to that required for the solution obtained by solving the model **P** using CPLEX [12]. Due to the $\mathcal{NP}$ hardness of **P**, large size problems (e.g. 100 storages and 3000 objects) cannot be solved using CPLEX within an acceptable time bound. We therefore resort to the alternate approach of calculating the accuracy using a lower bound of the problem **P**. We obtain the lower bound solution of the problem **P** by LP-relaxation (relaxing the binary constraint of the variable x_{ij} and declaring it a variable within $\{0,1\}$ range) and solving the LP-relaxed version of the problem **P**. We compute the percentage gap between the data movement obtained by heuristic and this lower bound as follows.

$$PercentageGap = \frac{Heuristic - LowerBound}{LowerBound} \times 100$$

We generated feasible problems by varying the size of database objects (j) uniformly within 1-100 MB. We varied the r_j for database objects following a Zipfian distribution (following typical notion of 20% of database objects are accessed in 80% of the cases) within 10-1000 bytes/sec. We varied the growth rate of database objects uniformly between 0-1000 KBytes/days. For baseline, we set the number of storage nodes to 100 and the number of objects to 200. Then, fixing storage nodes to baseline value (100), we varied the number of objects (500, 1000, 1500, 2000, 2500 and 3000). Similarly keeping the number of objects to the baseline value of 2000 we varied the number of storage nodes (50, 75, 100, 125, 150 and 175). We thus obtained 11 cases in total. The value of T is kept constant at 15 days (2 weeks) and the threshold value of utilization (U_{th}) is kept at 5%. For each of these cases, we generated 3 problem instances by varying the parameters as described above. In each case, the value of storage node sizes (B_i) and maximum bandwidth (U_i^m) are generated randomly within a upper bound and lower bound computed so that feasible solutions of the model exist. We then compute the average percentage gap in data movement for each case.

The variation of the *PercentageGap* with varying number of objects and varying number of storage nodes are shown in Figure 4 and 5 respectively. In both cases, the gap remains within 7% of the lower bound. This also implies that in cases where feasible solution exists, the heuristic solution lies within 7% of the optimal solution. Note, however, that here we compare against the lower bound solution which may be worse than the optimal solution. So the actual gap may be lower than those shown in the figures.

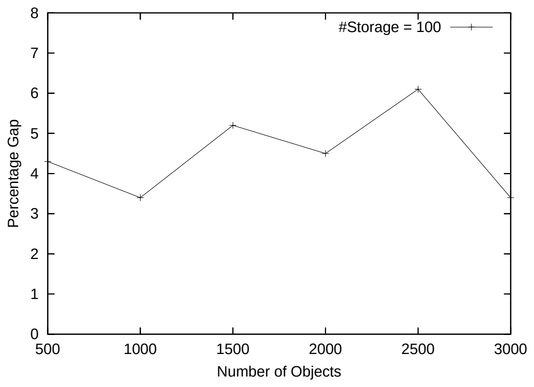

Figure 4. Varying number of objects.

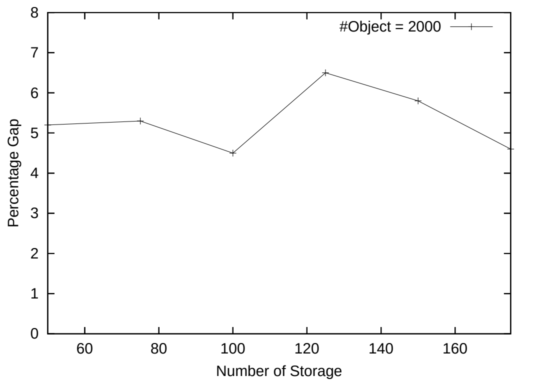

Figure 5. Varying number of storage nodes.

7.2 Convergence of Heuristic Algorithm

To demonstrate the convergence of the heuristic we plot in Figure 6 the value of $currentDeviation$ (as computed in the algorithm) in each iteration. Note how the deviation reduces in each successive iteration. The graph shows this deviation for three different values of number of objects 1000, 2000 and 3000; the number of storage nodes is kept constant at 50. The values of other parameters are generated as described in Section 7.1. The values of storage node sizes are generated based on feasibility of the solution. The value of maximum bandwidth of storage nodes are generated as a random number between 15-25 MBytes/sec in each of these three cases.

As can be seen from the Figure 6, the algorithm converges in all three cases within at most 5 iterations. In case of 1000 and 2000 objects the algorithm produces a feasible solution with a final deviation of zero. In case of 3000 objects the algorithm is unable to produce any feasible solution. however, unlike a model-based CPLEX approach that does not produce any solution in case of infeasibility, the heuristic actually reduces the deviation from 4500 to about 600 and stabilizes before exiting within 5 iterations. Thus our heuristic not only generates a solution that is best at any situation but also does so within a very small number of iterations. Further, the efficiency of the heuristic is excellent. The heuristic required approximately 5 seconds on an average for a run with 3000 objects and 175 storage nodes on a Pentium 2.4 GHz processor. We therefore believe that the heuristic is practical and can easily be used for online database storage management in a data center environment.

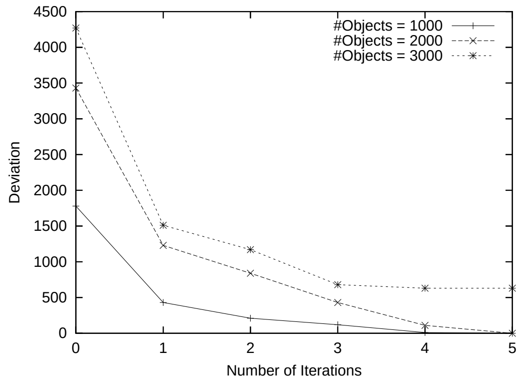

Figure 6. Convergence of heuristic algorithm.

8 Conclusion

In this paper we presented STORM, an automated approach that can guide effective utilization of storage nodes used for database storage, in a data center environment. We developed a math-programming model of the problem and showed that the the problem is $\mathcal{NP}$-hard. For the data center managers we developed a simple greedy heuristic that provides approximate solutions quickly. By simulation study we have shown that the greedy heuristic generates a solution for feasible problem that lies within 7% of the optimal solution. Further, even in cases where the actual formulation of the problem does not allow feasible solutions, the heuristic is still effective in significantly reducing the imbalance in bandwidth utilization across the storage nodes. The time-complexity of the heuristic algorithm is low order polynomial making it an efficient and practical solution for large number of database objects and storage nodes. In the future, we intend to extend our work to other storage systems such as mail systems and file systems.

Acknowledgements

This work was supported in part by the National Science Foundation under grant IIS-0534530 and the Department of Energy under grant DE-FG02-06ER25739.

References

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *Proc. ACM SIGMOD 1999*, pages 181–912, Philadelphia, PA, 1999.

[2] N. Allen. Don't waste your storage dollars: What you need to know. *Research note, Gartner Group*, March 2001.

[3] N. An, J. Jin, and A. Sivasubramaniam. Algorithms for index-assisted selectivity estimation. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):305 – 323, 2003.

[4] P. Aoki. Toward an accurate analysis of range queries on spatial data. *Proceedings of 15th International Conference on Data Engineering*, page 258, 1999.

[5] BMC Software. Capacity management and provisioning. www.bmc.com, 2005.

[6] P. Cappanera and M. Trubian. A local search based heuristic for the demand constrained multidimensional knapsack problem. *INFORMS Journal of Computing*, 17:82–98, 2005.

[7] P. Chu and J. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.

[8] Computer Associates. Storage management. www.ca.com/products, 2005.

[9] P. Furtado. Experimental evidence on partitioning in parallel data warehouses. In *DOLAP '04: Proceedings of the 7th ACM international workshop on Data warehousing and OLAP*, pages 23–30, New York, NY, USA, 2004. ACM Press.

[10] K. A. Hua and C. Lee. An adaptive data placement scheme for parallel database computer systems. In *VLDB '90: Proceedings of the 16th International Conference on Very Large Data Bases*, pages 493–506, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[11] IBM Inc. Storage area network (san). http://www-03.ibm.com/servers/storage/san/, 2006.

[12] Ilog Inc. ILOG CPLEX World's leading mathematical programming optimizers. http://www.ilog.com/products/cplex/, 2006.

[13] A. H. G. R. Kan, L. Stougie, and C. Vercellis. A class of generalized greedy algorithms for the multi-knapsack problemm. *Discrete Applied Mathematics*, 42:279–290, 1993.

[14] Karl Nagel & Co. Sarbanes-oxley. http://www.sarbanes-oxley.com/, 2006.

[15] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, January 2003.

[16] S. Khullar, Y. Kim, and Y. Wan. Algorithms for data migration with cloning. In *Proceedings of 22nd ACM Conference on Principal of Database Systems*, 2003.

[17] T. T. Kwan, R. McCrath, and D. A. Reed. NCSA's world wide web server: Design and performance. *IEEE Computer*, 28(11):68–74, 1995.

[18] E. Lamb. Hardware spending spatters. *Red Herring*, pages 32–33, June 2001.

[19] McDATA Corporation. Storage network extension and routing. http://www.mcdata.com/products/hardware/srouter/index.html, 2006.

[20] M. Mehta and D. J. DeWitt. Data placement in shared-nothing parallel database systems. *The VLDB Journal*, 6(1):53–72, 1997.

[21] Oracle Corporation. Automatic storage management technical overview: An oracle white paper. *Oracle Technology Network (http://www.oracle.com/technology/)*, November 2003.

[22] M. Stonebraker, P. Aoki, R. Devine, W. Litwin, and M. Olson. Mariposa: a new architecture for distributed data. In *Proceedings of 10th International Conference on Data Engineering*, pages 54–65, 1994.

[23] Veritas. Storage and server automation. Identify portions of the database that are not in use, 2005.