STORM: An Approach to Database Storage Management in Clustered Storage 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 in clustered storage environments is a manual, time-consuming, and error-prone task. Such management involves regular movement of database objects across nodes in the storage cluster 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, efficient analysis of gathered information, and effective decision-making for reconfiguring data layout. The reconfiguration process is guided by the primary optimization objective of minimizing the total data movement required for the reconfiguration, with the secondary constraints of space and balanced I/O bandwidth utilizations across the storage nodes in the cluster. We model the reconfiguration decision-making as a multi-constraint optimization problem which is $\mathcal{NP}$-hard. We then 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 nodes and N is the total number of database objects. 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 enterprises 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 [12] 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 [9]) - a network whose primary purpose is the transfer of data between computer systems and a cluster of storage nodes. Applications read from and write to a storage node in the cluster through SAN switches or routers [17].

Although SANs and storage clusters 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 storage nodes, both in terms of storage space and I/O bandwidth. Such skewed storage utilization eventually degrades the performance of applications and creates 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 nodes to avoid such skewness in the storage cluster. However, manual decision making in large data centers containing several terabytes of data and hundreds of storage nodes (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

[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, 16].

database elements (tables, indexes, logs, etc.) to the appropriate storage node in the cluster.

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 storage cluster environment and accordingly proposes a near optimal data movement strategy. Moving a large amount of data between storage nodes 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 clustered storage environment.

2. We present a mathematical model (which is $\mathcal{NP}$ hard) for the problem of balancing the I/O bandwidth utilization across the storage nodes in the storage cluster with the objective of minimizing data movement, given the storage node capacity constraints.

3. We 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 lower-bound solution. We make final remarks in Section 8.

2 Related Work

Research on data-placement in parallel database systems [8, 18, 7] 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. Our goal is to balance the utilization of shared storage nodes in a storage cluster across multiple database systems, and ensure the physical disk-space for future growth of database objects.

Distributed data storage systems such as Mariposa [20] have developed ways to place data in geographically distributed 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 clustered storage environment. Further, Mariposa optimizes for a WAN setting where network bandwidth is scarce and data are allocated based on optimal network usage. In our system database servers are connected to storage nodes over a high-speed network, which is typically the case in enterprise data centers. In such a scenario we can assume all storage nodes are equally accessible by database servers. We address the problem of balancing the storage node utilization, also allowing the future growth of database objects.

Storage management vendors such as Veritas [21], Computer Associates [6], and BMC [3] 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 utilized, are likely to result in suboptimal use of storage resources. Further, these solutions involve moving blocks of data from one storage node 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 [14] across storage nodes 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.

Oracle's Automatic Storage Manager (ASM) solution [19] 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 approach may fragment any table or index over the stripe size (which is a global setting), i.e. different parts of the same table or index may reside in different storage nodes. Our approach is general enough to be applicable not only in Oracle but also in most of the off-the-shelf database systems such as SQL-Server, MY-SQL and DB2. It can be applied in a heterogenous environment where a single data center has multiple heterogenous database systems sharing the same storage cluster, a typical case in enterprise data centers.

Finally, load balancing of server resource usage has been an active area of research for over a decade since early work on webservers [15]. 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.

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 cluster of storage nodes. Further, as typically the case, we assume that the data center comprises of several database server clusters; these clusters share the storage space provided by the storage node cluster.

Our work focuses on the interaction between the database servers and the cluster of storage nodes. Database servers allocate storage space in the storage nodes 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 a cluster of storage nodes in a database-centric system is to move data from one storage node to another to accommodate the future growth of objects and to balance the utilization of the individual node in the storage cluster. 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 storage node cluster 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 [13]. 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. Data center managers can choose to reconfigure the layout of objects to improve the storage utilization by using these decisions as hints.

Figure 1 depicts the architecture of STORM. Database

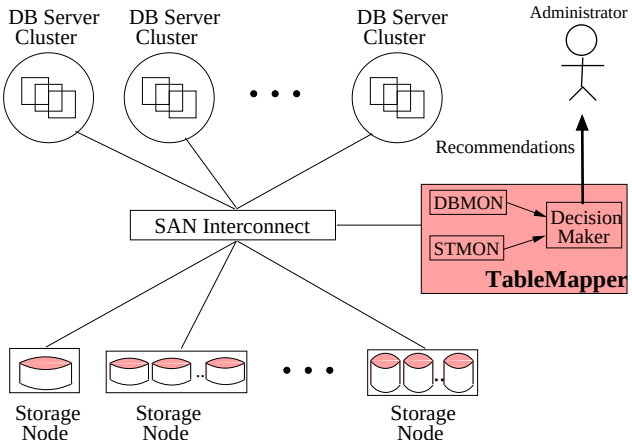

Figure 1. The STORM Architecture

servers are connected to storage nodes of the clustered storage environment through SAN interconnect (SAN router, switches etc.) The key component of our system is the *TableMapper*, which has access to the database servers as well as the cluster of storage nodes through SAN devices. 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 nodes such as storage capacity and I/O bandwidth. These storage data are mostly static and updated only when new storage is added or existing storage is taken out from the cluster. The storage utilization, i.e. which database object is using which storage node and how much space it is consuming, is dynamic information and is gathered by the STMON from the database servers. 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 Section 6. The decision maker analyzes this data and makes reconfiguration decisions (described 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, there are two key issues to consider. First, the TableMapper must be non-intrusive in collecting object and storage usage information from the database servers. Since this operation is performed only periodically, it can be performed during system idle time. The second challenge is to avoid a bottleneck at the TableMapper. 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 (for load distribution) to collect data from a large 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 database object layout on storage nodes. In doing so, it must balance multiple optimization objectives. First, the new configuration should be achievable with minimal data movement. We believe that this is the primary optimization objective due to the following reasons. Reducing the total amount of data movement will contribute to realizing the new configuration in less time and thus lower cost. It will reduce network traffic, and also the volume of data that could potentially be rendered unavailable during the move. These factors would also encourage data center IT managers to perform more frequent reconfiguration operations leading to reduced skewness in storage usage over time. Second, the new configuration must ensure that none of the storage nodes is overly utilized in terms of I/O bandwidth. This can be addressed by posing a secondary reconfiguration constraint so that the percentage I/O bandwidth utilization for each storage node is below the average percentage I/O bandwidth utilization across all storage nodes plus a small configurable threshold. Finally, the new configuration should support the future growth of database objects till the storage managers decide for another round of reconfiguration.

In the next section, we formally model and describe the dynamic storage reconfiguration decision-making problem and show that it is a hard problem to solve exactly. Subsequently, we present a heuristic algorithm (in Section 5) for such decision-making that provides good approximate solutions within an acceptable time bound.

4 Model

We describe the configuration decision making problem formally as *"given a set of database objects (J) with their present growth rate (g_j), usage characteristic (r_j) and size (s_j), given a set of storage nodes (I) with availble I/O bandwidth (U_i^m) and capacity (B_i) specifications and given the present assignment (c_{ij}) of database objects to storage nodes - determine a new assignment (x_{ij}) of objects to storage nodes that (i) will result in minimal physical movement of data across storage nodes 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 database objects for certain time (T)."*

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

Parameter	Description
i	Index for set of physical storage nodes (I)
j	Index for set of database objects, tables and indices (J)
c_{ij}	Equals 1, if j is currently located in storage node i 0, otherwise
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 node i in Bytes/sec
B_i	Storage capacity in bytes for storage node i
r_j	The average bytes/sec retrieved from object j to serve database requests related to object j
U_{th}	Toleration threshold in percentage for storage node over-utilization beyond 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 0, otherwise

Table 1. Model parameters.

Problem P:

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

subject to,

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

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

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

$$\sum_i x_{ij} = \sum_i c_{ij} \qquad \forall j \quad (5)$$

$$x_{ij} \in \{0,1\}, \qquad c_{ij} \in \{0,1\} \qquad \forall i,j$$

where **Z(P)** is the optimal value of the amount of data moved.

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

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

It can be shown 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)

[5, 4], which is known to be $\mathcal{NP}$-Hard. Details of the reduction are presented in [1].

Note that the above problem **P** is an integer programming (IP) problem. Due to computational complexity typically IP for large size problems (e.g., thousands of database objects and hundreds of storage nodes in a data center) are hard to solve using standard solvers like CPLEX [10]. Further the $\mathcal{NP}$-hardness of **P** makes it harder to obtain exact solutions. In the next section, we develop a simple heuristic algorithm that provides an approximate solution to the problem with an acceptable time complexity. Moreover, unlike CPLEX and the model based approach which do not provide any solution in case no exact solution to the problem exists, our heuristic algorithm will provide a solution that will reduce over-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 storage nodes), the heuristic aims at finding a new storage configuration that is better suited to serve the current request load.

Here we briefly describe the algorithm. The detailed psuedocode for the heuristic algorithm is given in [1]. 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}). Bootstrapping the system can be performed with a random assignment of objects to storage nodes.

The algorithm is a two-stage greedy algorithm. It first tries to move the smaller objects across storage nodes to achieve the objective goal before choosing to move the larger ones, i.e., greedy on size. In moving the objects it first tries 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. Greedy heuristics are known to give good heuristic solution for various kinds of knapsack problems [11]. A greedy heuristic also allows us to develop a simple algorithm that can be easily adapted by database administrators.[2]

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 a feasible solution (wherein the percentage I/O bandwidth utilization of all storage nodes are below that threshold value) does not exist, the algorithm terminates when the *deviations* (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}) in storage utilization in two consecutive iterations remain unchanged. Thereby, the heuristic still provides a solution that reduces (not nullify as in the case of a feasible solution) the over-utilization of I/O bandwidth (i.e. deviation) across the storage nodes.

Heuristic complexity. It can be shown that the total time complexity of the heuristic algorithm is $O(|J|log(\frac{|J|}{|I|} + (\frac{|J|}{|I|})^2)$.[3] Thus, our heuristic algorithm is low order polynomial and can be run quickly for a 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 inquiry for DBAs because they lend significant insight into how database objects should be placed in various storage nodes in the storage cluster. 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. Each profiling module is configured with a list of database servers to be monitored. For simplicity, we assume that the database servers have exclusive access to the storage nodes. For monitoring database access patterns, the DBMON submits a set of monitoring queries to the database servers, and processes the results to yield the monitoring information of interest. The STMON also queries the database servers and the storage nodes respectively to obtain dynamic information about space usage for individual database objects and static information such as storage capacity of individual storage nodes.

The data queried in the above monitoring process is not application data, but rather system meta-data. As a result, the total volume of data collected is small and the contention with application data access is minimal. Further,

[2] Due to space limitations, we elaborate the algorithm further elsewhere [1].

[3] [1] presents a detailed derivation, which we omit here due to space limitations.

the query interval is typically large and configurable; it can be made sensitive to database server availability (e.g., during off-peak hours or service down-time). Finally, once the query results have been received, all other processing takes place outside the system datapath.

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 Database Object Access Patterns

	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. Object-usage monitoring queries.

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. Q2 returns indexes and columns on which indexes are defined. Q3 returns a list of SQL statements executed on the database, and each statement's execution count. 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. Therefore, the query submission interval for Q3 must be chosen carefully. DBMON can determine database object usage statistics (base usage statistic # **1**) by aggregating execution counts.

Cache considerations. Modern database systems cache frequently accessed objects in memory. Table 3 shows the query and the system stored procedures in Oracle and SQL-servers respectively that provide specific statistics about cache hits. 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 (base usage statistic # **2**), 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.

6.2 Storage Consumption Patterns

	DBMS	Query Description	Query
Q4a	Oracle	Table-space 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 Filegroup	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 growth (base usage statistic # **3**). The query Q4 gives us the physical location of a database table or index i.e., 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 (base usage statistics **3**). Taking a series of these values over time provides data points which STMON uses to forecast growth of table size(g_j).

7 Simulation Results

In this section, we evaluate 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 problem

P using CPLEX [10]. Due to the $\mathcal{NP}$ hardness of **P**, large size problems (e.g. 100 storage nodes 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** using CPLEX. 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 (with 20% of database objects being accessed in 80% of the cases) within 10-1000 bytes/sec. We varied the growth rate of database objects uniformly between 0-1000 KB/day. 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.

The variation of the *PercentageGap* with varying number of objects and varying number of storage nodes are shown in Figures 2 and 3 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 lesser than those shown in the figures.

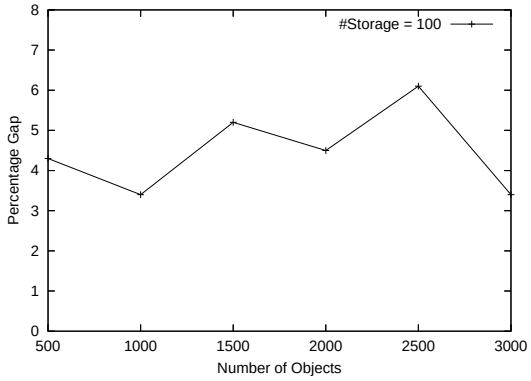

Figure 2. Varying number of objects.

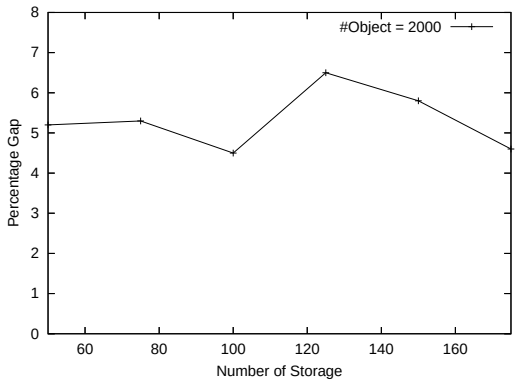

Figure 3. Varying number of storage nodes.

7.2 Convergence of Heuristic Algorithm

To demonstrate the convergence of the heuristic we plot in Figure 4 the value of current $deviation$ (as presented in section 5) in each iteration. Note how the deviation reduces in successive iterations. 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. Storage node sizes are generated based on feasibility of the solution. The value of maximum bandwidth of storage nodes are generated as random numbers between 15-25 MBytes/sec.

As can be seen from the Figure 4, the algorithm converges in all three cases within at most 5 iterations. In the case of 1000 and 2000 objects, the algorithm produces a feasible solution with a final deviation of zero. In the 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 near optimal 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.

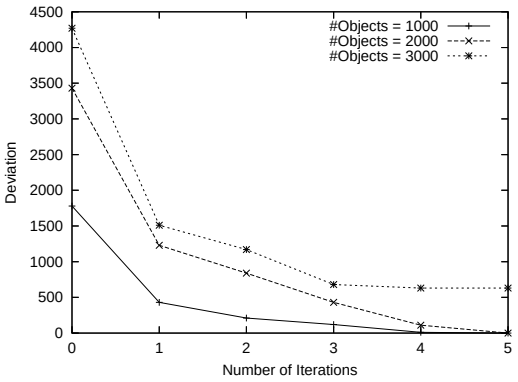

Figure 4. Convergence of heuristic algorithm.

8 Conclusion

In this paper we presented STORM, an automated approach that guides effective utilization of nodes in a storage cluster used for database storage. We presented automated techniques that can monitor the utilization of both storage nodes as well as database objects such as tables or indexes, while incurring low overhead. We developed a math-programming model of the decision-making problem of optimal layout reconfiguration, which turns out to be $\mathcal{NP}$-hard. For database administrators, we developed a simple two-stage greedy heuristic that provides approximate solutions quickly. Using a 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 solution, 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 servers and file systems.

9 Acknowledgements

This work was supported by NSF grants IIS-0534530 and HRD-0317692, and a Department of Energy grant No. DE-FG02-06ER25739.

References

[1] STORM: An Approach to Database Storage Management in Data Center Environments. http://www.cs.fiu.edu/ raju/WWW/publications/techreports/TR-2006-09-02.pdf, September 2006.

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

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

[4] 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.

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

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

[7] 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.

[8] 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.

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

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

[11] 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.

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

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

[14] 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.

[15] 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.

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

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

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

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

[20] 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.

[21] Veritas. Storage and server automation. http://www.symantec.com/Products/enterprise, 2005.