Management of Highly Dynamic Multidimensional Data in a Cluster of Workstations

Vassil Kriakov[1], Alex Delis[2], and George Kollios[3]

[1] Polytechnic University, Brooklyn NY 11201
vassil@milos.poly.edu
[2] The Univ. of Athens, Athens GR15771, Greece
ad@di.uoa.gr
[3] Boston University, Boston, MA 02215
gkollios@cs.bu.edu

Abstract. Due to the proliferation and widespread use of mobile devices and satellite based sensors there has been increased interest in storing and managing spatio-temporal and sensory data. It has been recognized that centralized and monolithic index structures are not scalable enough to address the highly dynamic nature (high update rates) and the unpredictable access patterns in such datasets. In this paper, we propose an adaptive networked index method designed to address the above challenges. Our method not only facilitates fast query and update response times via dynamic data partitioning but is also able to self-tune highly loaded sites. Our contributions consist of techniques that offer dynamic load balancing of computing sites, non-disruptive on-the-fly addition/removal of storing sites, distributed collaborative decision making for the self-administering of the manager, and statistics-based data reorganization. These features are incorporated into a distributed software layer prototype used to evaluate the design choices made. Our experimentation compares the performance of a baseline configuration with our multi-site system, examines the attained speed-up as a function of the sites participating, investigates the effect of data reorganization on query/update response times, asserts the effectiveness of our proposed dynamic load balancing method, and examines the behavior of the system under diverse types of multi-dimensional data.

Index Terms: Data Management in Cluster of Workstations, Networked Storage Manager, Self-tuning Storage Nodes, and Multi-dimensional Data.

1 Introduction

Modern applications have to manage continuously growing and morphing volumes of data [2, 19, 9, 26, 7]. The high update rates and unpredictable access patterns in such applications make it challenging to provide short and consistent database response times. For example, the Terra spacecraft (EOSDIS project [7]) produces around 200 GB/day and Landsat 7 another 150 GB/day of geophysical data [30]. As pointed out in [29], *science is becoming very data intensive* for many fields. A wide range of integrated medical instrumentation and patient-care systems also produce massive spatio-temporal data [5, 24, 23]. The management of data networks and content delivery networks calls for efficient data visualization of network datasets [1] to help track changes and maintain good levels of resource provisioning for applications. Finally, critical areas that involve continuously changing and voluminous spatio-temporal data include intelligent transportation and traffic systems, fleet and movement-aware information systems, and management of digital battlefields. The inherent multi-dimensional nature of this data calls for the use of indexing methods that are capable of providing efficient data access [21, 25, 22]. It is worth pointing out that in the aforementioned areas data access as well as update patterns vary over time due to a number of reasons including ever-changing user interests, weather conditions, formation of new traffic congestion points, production of updated medical records, sensor failures, and network topology changes.

In order to facilitate the continuous, yet incremental growth of data without resorting to specialized hardware, we develop a networked storage manager based on a Cluster of Workstations (COW) connected via a high speed LAN. Portions of data are assigned to and indexed at these workstations (sites). We use R*-trees to index multidimensional data [9, 4] because their leaf-level nodes are not correlated (in contrast, there is an absolute order of the leaves of a B$^+$-tree). This feature is leveraged by our system to extract a subset of the data indexed by one of the sites in the COW, insert it in the R*-tree of another site, and preserve the overall integrity of the dataset [18]. This load balancing through data migration involves a number of challenging trade-off questions: should data be moved at all, which data should be migrated, how much data is it necessary to ship and finally, between which sites is data to be migrated. We resolve these questions by adopting soft lower/upper limits on load variations, maintaining access statistics for nodes in the R*-trees, and continually controlling the load of the COW sites.

Our proposal builds on prior research [18, 20, 27, 28]. However, a number of salient features substantially differentiate our work and include: support for high update rates; decentralized collaborative decision making to improve scalability; hot spots identification for efficient load balancing; graceful upscaling without any down-time; and lastly, evaluation of the usefulness of a Top-k-Levelvariable indexing scheme in the COW environment. We develop a full-fledged prototype in C++/BSD-sockets and carry out an extensive experimental study to demonstrate the benefits of our proposed techniques. Our main performance indicator is the average response time (ART) of requests (queries or updates) [8]. The main results of our evaluation are: a) achieved speedups of up to 50 times as compared to identical non-self-tuning systems (eg. [28]); b) sizable (10-50%) concurrent updates of the data set impose only minimal degradation of the average query response times; c) robust scalability characteristics are exhibited with minimal human intervention; d) the proposed Top-k-Level indexing scheme establishes that query redirection is best achieved through broadcasting.

The remainder of this paper is structured as follows: Section 2 discusses related work. Section 3 describes the architecture of our system and outlines the proposed load-sharing and data migration techniques. Our experimental analyses are discussed in Section 4 while conclusions and future research directions can be found in Section 5.

2 Related Work

In [6, 16] distributed extendible and linear hashing are examined. A combined distributed index-hashing approach for one-dimensional data is proposed in [10]. Indexing suitable for shared-memory multiprocessor systems appears in [17], while [3] discusses issues pertinent to the reliability of distributed structures. [11] introduce the B-link tree which provides multiple levels of parallelism for accessing one-dimensional data. The levels of parallelism are achieved by a shared-nothing distributed approach, locking mechanisms working off individual sites, and partial replication of data. A load-conscious approach is also proposed in [14]. Load balancing techniques for parallel disks that allow for judicious file allocation and dynamic redistributions when page access patterns change are discussed in [27].

In [13] a "semi-distributed" version of R-Trees is proposed and formulae regarding optimality of data sizes and response times are derived. In [12] parallelism is exploited by distributing an R-Tree across several disks managed by a single processor and in [28] this concept is extended to a shared-nothing R-tree architecture. For one dimensional dataset, a globally height-balanced adaptive parallel B$^+$-tree (AB$^+$-tree) is introduced in [15]. An improved version based on R-trees is proposed in [18], where the strength of the approach is evaluated via a simulation study. Finally, on-line reorganization of a centralized B+-tree is investigated in [31].

Our proposal and development work introduce a number of innovations including: a) a dynamic load balancing component facilitates data reorganization among the distributed computing sites due to random and heavily mixed workloads; b) we use on-the-fly fine-tuning of data distributions to tailor for high-rate access patterns and frequently occurring sizable updates; c) data selection during migration occurs in a way that minimizes the amount of data shipped, while maximizing the improvement on performance. The scalable LAN-based architecture reaps the benefits of a centralized global view at a master site, without

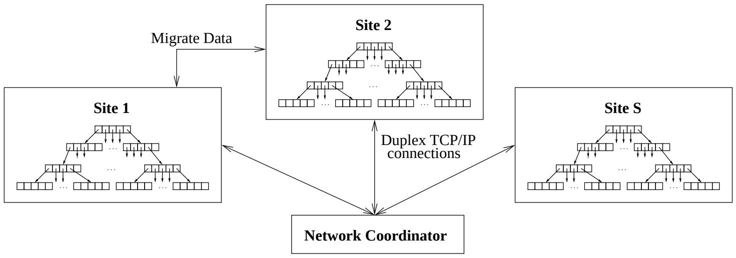

Fig. 1. Logical Architecture

impeding scalability. System upscaling can occur on demand, without any downtime, by simply adding more COW client sites which are gracefully populated.

3 System Architecture, Models and Heuristics

The ultimate design goal of our COW-based manager is for it to exhibit superior performance under very intensive workloads, where skewed access patterns shift load conditions, and sustainable update rates increase resource demands.

3.1 Distributed Storage Manager Architecture

Our model consists of a cluster of workstations (COW) which communicate over a high-speed network and host the underlying data set. The COW storage manager functionality is divided between the the clients and a network coordinator (or simply coordinator) as depicted in Figure 1. The responsibilities of the network coordinator are reduced to a minimum to eliminate any bottleneck effects. The coordinator keeps a global load table which is used when making load balancing decisions. Any of the sites can also act as a coordinator.

Each site's basic tool for autonomous data management is as an R*-tree which indexes the local data set fragment assigned to it. Furthermore, each client receives and executes requests which are submitted locally (through APIs), forwarded from other clients, or forwarded from the coordinator. The supported request operations are the containment or intersection queries and data insertions and deletions (updates).

In the following sections we look more closely at the reasons for some of our design choices and describe the client-client and client-coordinator interactions.

3.2 Evaluation of Top-k-Level Indexing

Past research efforts propose that a server holds a "distribution catalog" which contains partial information about the data located at the client sites. In [28,

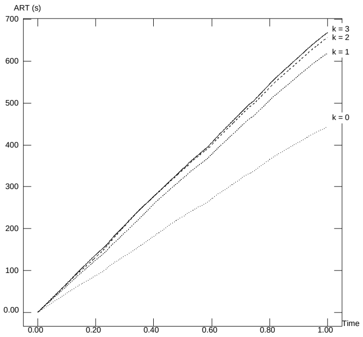

Fig. 2. Evaluating the costs of maintaining a central distribution catalog: response time is seriously affected under high update rates.

13] the coordinator holds copies of all non-leaf-level nodes of the index structure of each client, with the postulation that this portion of the index comprises a negligible part of the total index space. This approach suffers under heavy and frequent updates as these trees have to be modified continually. In an ad-hoc manner, [18] suggests that only the root level nodes of each client site should be held at the coordinator. Empirically, we establish that the wide overlap in root nodes renders this method inefficient.

A more general approach is to keep the Top-k-Levels for each tree in the coordinator. We conduct a series of experiments to evaluate this approach for values of k ranging from 0 (no catalog) to $TreeHeight - 1$ (full catalog). Figure 2 shows the results for an experiment involving ten million elements distributed among six sites where one million insertions are intermixed with a sustained high workload of 200,000 queries each retrieving 0.1-1% of the dataset. Results for other experiments are similar but are omitted for brevity. The y-axis represents the average response time (ART), which is our primary performance measure. Our experimental results show that, under conditions involving heavy update loads, the best performance is achieved when Top-k-Level Indexing is not used and requests are broadcast to all client sites (i.e. $k = 0$). Based on this finding and under the assumption of heavy update workloads, we adopt the approach where the coordinator holds no information on data placement at client sites. Another significant benefit of this scheme is that any site may be an entry point of data requests which reduces the centralized role of the coordinator and, consequently, improves on scalability.

3.3 Self-Tuning Principles

Dynamic load balancing is required in a COW environment subject to changing access patterns as it helps achieve the following goals:

1. Detection of hot spots due to skewed access patterns and redistribution of loads without service disruption and performance degradation.
2. The overhead of self-tuning is more than compensated for by the resultant performance gains after completion of balancing.
3. No administrative work is involved during the redistribution process.

To our knowledge, this is the first work to address this issue in the described shared-nothing environment. In [27], the matter is discussed in a shared-memory parallel-disk system where device statistics are readily available. The data migration algorithms in [18] do not identify how queries are handled during the redistribution process and do not deal extensively with the issue of skewed access patterns on a per-site basis.

In our network storage manager, dynamic load balancing is facilitated through on-line data reorganization. To avoid disruptions to user requests, we employ a concurrency control mechanism between the client sites involved in the data migration. The overhead of self-tuning is minimized through quick but careful selection of data for migration such that the balance achieved is near-optimal, while the amount of data migrated is minimal. In addition, we employ a distributed collaborative decision making process during the load balancing phase which reduces processing at the network coordinator and minimizes the number of load-balancing considerations. These points are discussed in detail in the following sections.

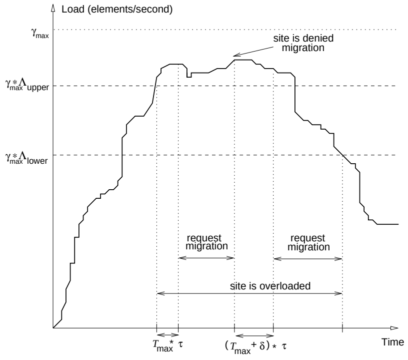

Fig. 3. Sample load variations for a given client site.

Parameter	Symbol	Values
load capacity	γ	3k - 30k
max load	γ_{max}	3k - 30k
load pct	λ	γ/γ_{max}
sampling period	τ	1 second
max samples	T_{max}	3 - 5
load increment	δ	1 - 3
upper threshold	Λ_{upper}	50-85%
lower threshold	Λ_{lower}	40-75%
load deviation	Δ	5-25%
sites	N	1 - 20
dataset size	D	0.1M - 10M

Fig. 4. System parameters and values for individual sites.

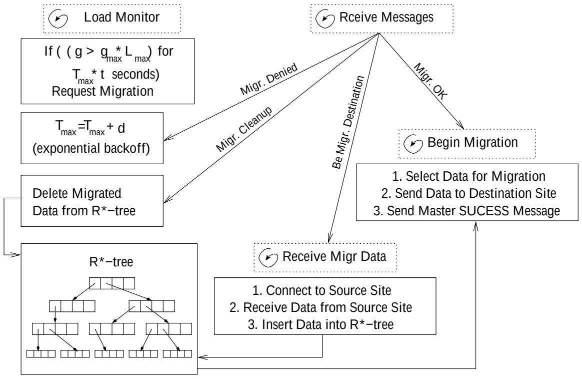

Fig. 5. Overloaded client sites request permission to perform migration. If the request is denied, T_{max} is incremented and no migration occurs. If the request is granted, data is shipped to the recipient site

3.4 Component Interaction

We define "client site load" as the number of data elements retrieved per second by a client site. This measurement is derived in connection with the CPU usage, disk I/O, and memory paging operations and implicitly reflects a workstation's resource utilization. In our experiments this value ranged from 3,000 to 30,000 elements per second (see table in Figure 4).

Each client i continuously measures its current load γ^i for each epoch elapsed (a sampling period of 1 second as indicated in the table in Figure 4). Clients have a fixed maximum sustainable load capacity γ^i_{max}, which can be determined a-priori by running a test benchmark[4] and is used to compute the current load percentage $\lambda^i = \gamma^i / \gamma^i_{max}$. Clients use the two system-wide parameters Λ_{upper} and Λ_{lower} to determine whether they are overloaded. Site i is considered overloaded as long as the condition

$$(\gamma^i > \gamma^i_{max} * \Lambda_{upper})$$

holds true for an epoch. Using this epoch prevents load balancing from occurring during spurious high loads. Prior to a client site's first migration request, the epoch must last at least $T_{max} * \tau$ seconds, where τ is the load measurement time interval in seconds, and T_{max} is the number of load measurements. If a site requests migration but the coordinator decides that the system is balanced and denies the request, the site increases its value of T_{max} by δ. T_{max} is reset to its original value when a migration request is granted. When a client considers itself to be overloaded it sends a *Request Migration* message to the network

[4] To establish the value of γ^i_{max} for a site i, we retrieve all data stored at that site. Assuming that sufficient amount of data is present, the site operates at its maximum sustainable load γ^i_{max} as reported by the *Load Monitor*.

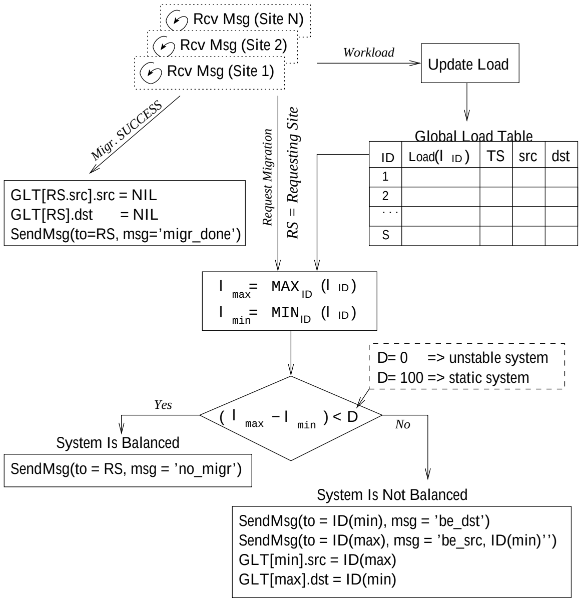

Fig. 6. The coordinator qualifies data migration based on the current load of each site, and selects for destination sites with the least load.

coordinator as indicated in Figure 5. This triggers the self-tuning mechanism and the coordinator evaluates the system's state of balance as shown in Figure 6. It is important to note that there is no continuous processing or polling at the coordinator. This certainly aids in the scalability of our architecture.

Effectively, the set of measurements and parameters in the table in Figure 4 provides soft thresholds for determining a site's load state. This reduces the number of migration requests during system-wide overloads when self-tuning is not possible. In essence, the dynamic tuning of T_{max} allows client sites to "learn" about the overall state of the system and attempt to adjust accordingly. A sample load situation for a client site is given in Figure 3, where it can be seen that the client requests migration only after its load has crossed $\gamma^i_{max} * \Lambda_{upper}$ for $T_{max} * \tau$ seconds, and discontinues its migration requests after its load line crosses $\gamma^i_{max} * \Lambda_{lower}$.

The coordinator records each site's load in the Global Load Table (GLT) shown in Figure 6 and uses this information to decide whether the COW manager is balanced. The network coordinator computes the difference between the loads of the most (λ_{max}) and least (λ_{min}) loaded sites in $O(N)$ time where N is the number of active client sites. When the condition:

$$(\lambda_{max} - \lambda_{min}) < \Delta$$

qualifies, the system is balanced. The parameter $\varDelta$ constitutes the system's tolerance for imbalance in terms of percentage and can be configured according to the specific application needs. Finding an optimal value for $\varDelta$ is beyond the scope of this paper, but we provide an empirical approach and experiment with values (5% - 25%) that are deemed representative for the parameter.

When the system is balanced, the requesting site is denied migration. Otherwise, the least loaded client is selected to be the destination site and the requesting site is redirected to continue negotiations with that client. The details of these negotiations are discussed in the next section. Since concurrent requests for migration may be issued by multiple client sites, the coordinator marks current destination/source pairs in the GLT, indicated by the 'dst' and 'src' columns in Figure 6. Such pairs of clients are not considered for destination candidates until the migration process between them completes.

3.5 Data Migration

The data migration scheme must be very efficient: data must be selected quickly and it must be shipped to the recipient site fast. To achieve these goals, each site collects access and update statistics for each node in its R*-tree tree. This information helps select minimal amount of data for migration, while maximizing the effect on load redistribution. This reduces the overhead of data transfers among the client sites and increases the system's self-tuning responsiveness.

In the context of skewed access patterns following Zipfian or Gaussian distribution we maintain that it is of significant importance *what* data is claimed for migration. If data accesses are uniformly distributed in space, to achieve a desired load reduction, say 50%, a client has to migrate an equivalent proportion (50%) of data. When access patterns are skewed, the degree of skew determines the amount of data to be migrated. Figure 7 depicts the relationship between load reduction rates and data migration size for various types of access skew. It can be seen that for a desired load reduction, very few elements must be redistributed under higher skews as compared to a uniform access distribution. Thus, by exploiting access pattern information, migration overheads can be reduced substantially.

Prior to selecting migration data, the overloaded client determines a target load reduction λ_{target}. This is the equilibrium point between the local load and the destination site's load: $\lambda_{target} = (\lambda_{src} - \lambda'_{dst})/2$. λ_{target} represents the load percentage that the overloaded site would like to reduce its load by while λ'_{dst} is λ_{dst} normalized to the source's maximum load capacity γ^{src}_{max}. This normalization is necessary when the COW is composed of heterogeneous sites with different capacities γ_{max}.

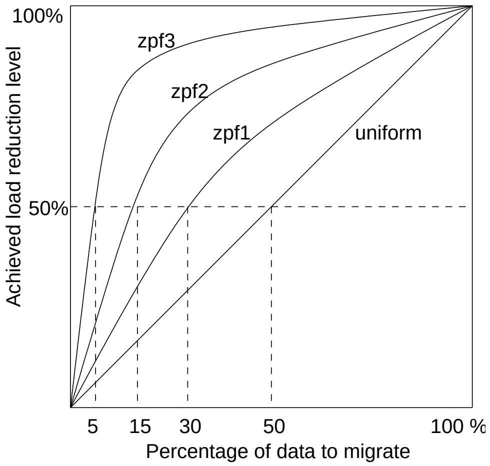

Fig. 7. With skewed access patterns, the amount of data that must be migrated to achieve a desired load reduction sharply decreases as compared to uniform access distributions. Therefore it is important to identified skewed access patterns when dealing with data redistribution.

To select data for migration, a client first examines its root in the R*-tree and using its access statistics determines the total R*-tree load over all subtrees st:

$$TotalLoad = \sum_{st\ \epsilon\ root} (\alpha * st.read\_count + \beta * st.write\_count)/\Delta t$$

where α and β are weight coefficients for adjusting the significance of reads relative to writes since writes are usually more costly. The $TotalLoad$ represents the frequency of reads and writes applied to the tree.

Consequently, the most loaded subtree at the root is discovered. If this subtree's load is not sufficient to reach the target load reduction (i.e. if *MaxLoadedSubtreeLoad / TotalLoad* $< \lambda_{target}\%$), the subtree is selected for migration to the destination site and the selection process terminates. Otherwise, that subtree is examined: its most loaded subtree is found and the evaluation process is repeated recursively. Note that the $TotalLoad$ is determined only at the root level. If the leaf-level is reached and no subtree has been selected for migration, the most loaded node in that leaf is selected for migration. To select data for migration, a site navigates its R*-tree tree, following the most loaded branches. Therefore, the data selection process runs in time proportional to the height of the tree, which is logarithmic with the number of elements indexed.

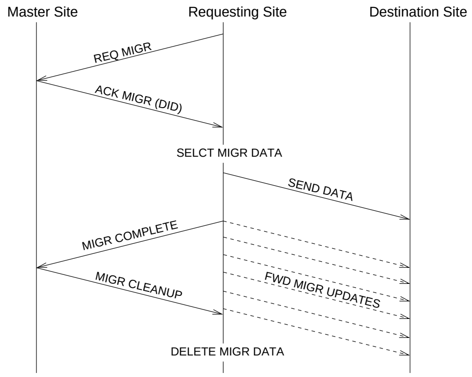

Fig. 8. Concurrency Control

3.6 Concurrency Control and Upscaling of the COW Manager

Once a subtree is selected for migration, a concurrency mechanism allows for data migration and query execution to proceed simultaneously. This concurrent execution is important not only from a synchronization standpoint but also because of the overheads involved during data reorganization. Thus, we allow request processing and data migration to occur simultaneously. To facilitate this, the subtree chosen for migration is marked prior to any data migration. Queries are allowed to enter the marked subtree at no cost. When updates enter a marked subtree, they are propagated to the destination site which received the migrated data as Figure 8 shows.

Once migration commences, two copies of the data exist: one at the destination site and one at the source site. The forwarding of updates guarantees that the two sets are synchronized. When migration is completed, the requesting site sends a “Migration Complete” message to the coordinator. The coordinator immediately enqueues a “Migration Cleanup” message on the outgoing queue for the requesting site. When the requesting site receives this message, it safely deletes the migrated data and stops forwarding messages to the destination site. At this stage, the data migration process is complete and there is only one copy of the migrated data at the destination site.

Due to this concurrency control mechanism, our COW manager provides a flexible environment for automatic system upscaling with the growth of the data set. New sites can be introduced into the distributed system by attaching them to the same network cluster. Gradually, part of the data set is relocated to the new site, alleviating the workload on the other sites. Down-scaling of the system is accomplished in a similar manner.

4 Experimental Results

4.1 Experimental Setup

Our prototype system is entirely developed in C. The computing sites consist of a cluster of SPARC-stations connected through a dedicated high-speed (100Mbps) LAN.

The experiments were performed using both synthetically generated and real-life data. The synthetic data was either uniformly distributed (hyper-squares with sides of length 0.0005) or exponentially distributed (using Gaussian distribution with $\sigma = 1.0$ and $\mu = 0$) in unit space. Furthermore, the experiments were performed with data of three dimensionalities: 2D, 4D, and 8D. The number of synthetic data elements (rectangles and hyper-cubes) used in the experiments ranged from one-hundred thousand to ten million. The queries were also synthetically generated (hyper-) rectangles, distributed uniformly or exponentially in the unit square. Three types of queries were used with sides of length 0.005 (small), 0.015 (medium), and 0.05 (large). The main factors affecting the experiment's workload are the query area and the query inter-arrival rate. The real-life data[5] consisted of 208,688 points in two-dimensional space representing functional densities of a computational-fluid-dynamics experiment on the wing of an airplane (Boeing 737).

Our main performance metric is the turnaround time for a given query (measured in seconds). We refer to this as the average response time (ART). The experimental objective was to evaluate the performance of the COW-based self-tuning manager under skewed access patterns and compare it to a system which does not employ dynamic load balancing. By "skewed access patterns" we refer to accesses which create hot spots in the distribute COW-manager. For persistently high workloads, speed-ups of a factor of 50 were achieved. Under short peak workloads, the performance improvements were a factor of 5.

Moreover, we evaluated experimentally the benefits of the combination of the dynamic load balancing algorithm with our selective data migration technique. Although the technique incurs extra overhead on the R*-tree processing, it resulted in migration of up to 3 times fewer data elements as compared to when no access information was tracked.

4.2 Results

Multi-Site Gains: Figure 9 depicts the performance gains achieved by increasing the parallelism of the COW-based manager by introducing new sites. For the cases when one to four sites were used, the system's query processing time was

[5] This and additional multi-dimensional data can be found at Scott Leutenegger's web page: http://www.cs.du.edu/~leut/MultiDimData.html

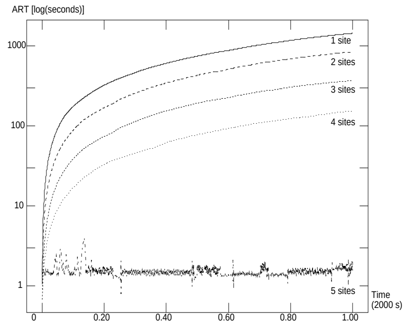

Fig. 9. Response times of a static system with 1 million 2-D elements for varying number of sites (log scale).

slower than the query arrival time. When a fifth site was introduced, the system was able to deliver optimal performance for the specific dataset, and the response times were only affected by the processing of each query at the five sites (since queries did not wait on the queue for previously submitted queries to complete). On a multi-site system there is an additional improvement for insertions and updates due to the smaller amount of data managed at each site (approximately n/N records, as opposed to n records on a single site system). This reduces some of the processing costs for data management.

Self Tuning Improvements: Figure 10 shows the response times for a two site system indexing 100,000 two-dimensional rectangles. During the experiment 10% of the stored data was modified. For the given workload, which was skewed to retrieve 70% of the requested data from one of the two sites and the remaining data from the other site, the system could not cope with the requests when self-tuning was turned off. Because the data retrieval time was greater than the query inter-arrival rate, the response time of the system increased progressively. However, with dynamic load balancing, during the first one-third of the experiment, the system automatically migrated data from the overloaded site to the underloaded site, resulting in significant improvements (up to a factor of 25).

The loads of the two sites for the duration of the experiment with migration are shown in Figure 11. The skewed query workloads cause the loads on the two sites to differ by about 50%. This is a typical unbalanced system. When the overloaded site requests migration permission from the coordinator, it is granted the request and the site begins migrating data to the underloaded site. This is seen in the gradual increase of the load of the underloaded site as it begins to index more and more data. When the two loads converge (at the 50% mark) data migration ceases and the system is optimally balanced. This corresponds to the steady low response time in Figure 10.

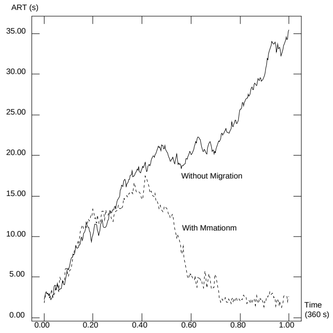

Fig. 10. 100k 2-D Expo. records, 10k updates on 2 sites (Speed-up: 5.12)

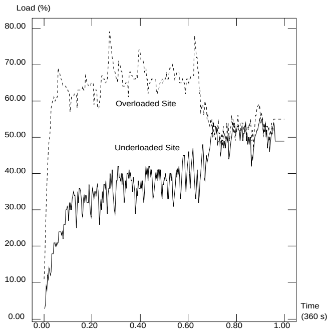

Fig. 11. Data migration effect on site load: gradual load balancing

Similar results were achieved with the real-life data set although the results are not shown due to space limitations. Surprisingly, the performance improvements were higher when operating on the real-life data: the speed-up was a factor of 9.23 on average.

With uniformly distributed data set (Figure 12) data migration again resulted in overall better performance. However, the gain ratio was much lower. This was due to two factors. Firstly, in general the response times for the uniformly distributed data set were much shorter than in the previous two cases (5.5 seconds vs. 35 seconds with exponential and 110 seconds with real-life data). Second, the overhead of data migration in the first half of the experiment was higher than the compensation achieved by moving the data to the overloaded site. However, the eventual balancing out of the load between the two sites resulted in a steady low response time (approximately 2 seconds).

The results of dynamic load balancing were comparable when data of higher dimensionalities was indexed as can be seen in Figures 13 and 14. With self-tuning, the system was slower at balancing out the load among the two sites than in the case of two-dimensional data. The spikes of the graph in the self-tuning case are result of the occurring data migration–the periods when the site is busy selecting data for migration, shipping it to the destination site, and removing it locally. These are periods when incoming requests are not processed, hence the short spikes.

When more sites are available (Figures 18, and 15), it takes more time to balance the system load because the dataset size is larger. For example, Figure 15 shows the response times of a system with ten client sites, two of which are overloaded (processing 70% of the workload) and eight underloaded (processing the remaining 30% of the workload). In such environment, it took almost 20

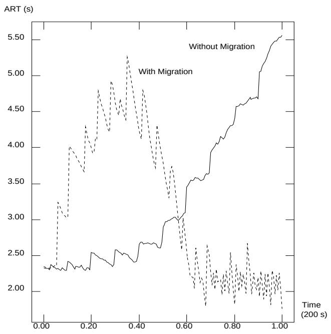

Fig. 12. 100k 2-D Uni. records, 10k updates, 2 sites (Speed-up: avg. 1.23, max. 3.14)

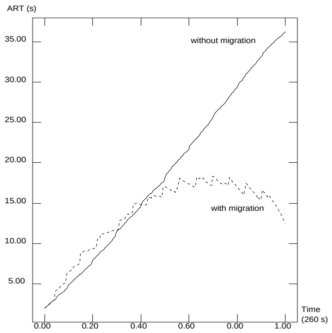

Fig. 13. 100k 4-D Uni. records, 10k updates, 2 sites (Speed-up: avg. 1.29, max. 2.88)

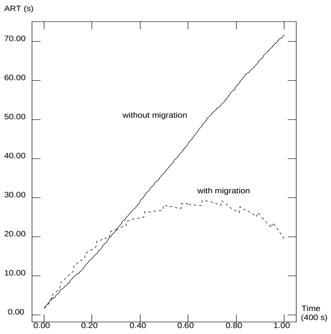

Fig. 14. 100k 8-D Uni. records, 10k updates, 2 sites (Speed-up: avg. 1.85, max. 3.99)

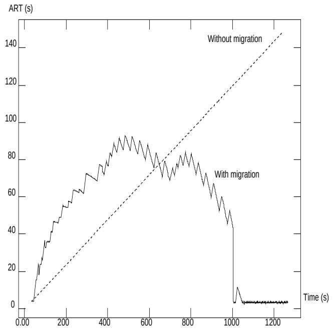

Fig. 15. 3M records, 0.5M updates, 10 sites, speed-up: 8.86

minutes for the system to achieve a balanced state. as larger amounts of data are stored on more sites, the "migration spikes" become more prominent, resulting in larger delays. However, it is important to note that when the system does reach a balanced state, the query response times decrease significantly, whereas in the case of no data migration, there is no prospect for such a decrease unless the query workload decreases.

In the case of system upscaling Figure 16 depicts the load percentage of one of the sites of a working system where at time 50 seconds, a new site was introduced in the system. At that point, load balancing was initiated and through a series of data migrations, the load on the specific site was more than halved as its data was relocated to the newly introduced site.

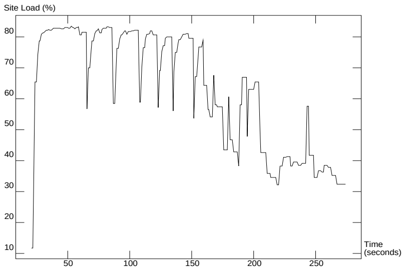

Fig. 16. System upscaling: a new site introduced at t = 50 seconds.

area (mu^2)	No. queries	delay (μs)	avg. gain
0.025	100000	2000	0.83
0.225	36000	3000	5.16
0.025	60000	2000	0.97
0.225	36000	3000	5.38
0.025	60000	2000	0.88

Fig. 17. Query workload distribution for peak workloads.

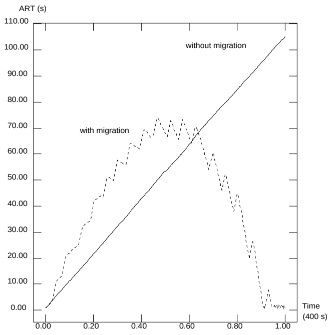

Fig. 18. 500k 8-D Uni. records, 50k updates, 4 sites. (Speed-up: avg. 7.23, max. 153.73)

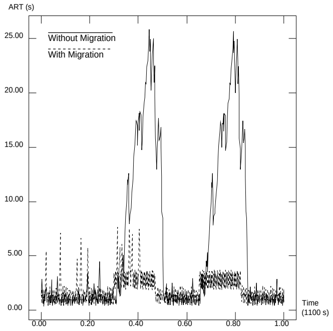

Fig. 19. Effect of migration during normal vs. peak workloads.

Previously discussed experiments were performed under constant workload conditions. To evaluate the COW-based manager's performance under more realistic conditions we created a workload consisting of relatively infrequent requests (see Table 17 for details on the workload), with two instances of high workloads in the duration of the experiment. Under normal workloads, the self-tuning system under-performed due to the occurring data migrations. However, the load balancing paid off handsomely during peak loads (see Figure 19), where the self-tuning system performed up to 8 times better.

5 Conclusions and Future Work

We have presented a self-tuning storage management system for multi-dimensional data distributed on a cluster of commodity workstations. Compared to prior approaches, our network-based system exhibits significant performance improvements due to a number of techniques: 1) dynamic data reorganization for load balancing identifies hot spots in the COW to efficiently manage data migrations; 2) distributed collaboration in the self-tuning decision process avoids the bottleneck of the central site; 3) a variable-level distribution catalog reveals the system's scalability issues when managing high volumes of frequently updated data. The above provide for robustness and consistent performance in a variety of settings, especially in conditions of unpredictably changing access patterns and high frequency updates to the underlying data. Our prototype helped us to evaluate empirically the key features of our proposal and the trade-offs of our system design. Future work includes extension of the prototype to incorporate bulk-loading mechanisms for data migration, replication schemes, and a server-less architecture.

References

1. J. Abello and J. Korn. MGV: A System for Visualizing Massive Multidigraphs. *IEEE Trans. on Visualization and Computer Graphics*, 8(1), Jan-March 2002.
2. T. Barclay, D. Slutz, and J. Gray. TerraServer: A Spatial Data Warehouse. In *Proc. of ACM SIGMOD*, pages 307–318, 2000.
3. F. Bastani, S. Iyengar, and I. Yen. Concurrent Maintenance of Data Structures in a Distributed Environment. *The Computer Journal*, 31(12):165–174, 1988.
4. N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proc. of ACM SIGMOD 1990*, pages 322–331. ACM Press, 1990.
5. V. Calhoun, T. Adali, and G. Pearlson. (Non)stationarity of Temporal Dynamics in fMRI. In *21st Annual Conference of Engineering in Medicine and Biology*, volume 2, Atlanta, GA, October 1999.
6. C. Ellis. Distributed Data Structures: A Case Study. *IEEE Transactions on Computers*, 34(12):1178–1185, 1985.
7. The Earth Observing System Data and Information System. http://spsosun.gsfc.nasa.gov/eosinfo/EOSDIS_Site/ index.html.
8. J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan–Kaufman, San Mateo, CA, 1992.
9. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.
10. T. Honishi, T. Satoh, and U. Inoue. An Index Structure for Parallel Database Processing. In *IEEE Second International Workshop on Research Issues on Data Engineering*, pages 224–225, 1992.
11. T. Johnson, P. Krishna, and A. Colbrook. Distributed Indices for Accessing Distributed Data. In *IEEE Symposium on Mass Storage Systems (MSS '93)*, pages 199–208, Los Alamitos, Ca., USA, April 1993. IEEE Computer Society Press.
12. I. Kamel and C. Faloutsos. Parallel R-trees. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, June 2-5, 1992*, pages 195–204. ACM Press, 1992.
13. N. Koudas, C. Faloutsos, and I. Kamel. Declustering Spatial Databases on a Multi-Computer Architecture. In *Advances in Database Technology - EDBT'96, 5th International Conference on Extending Database Technology, Avignon, France, March 25-29, 1996, Proceedings.*

14. B. Kroll and P. Widmayer. Distributing a Search Structure Among a Growing Number of Processors. In *Proceedings of the 1994 ACM SIGMOD Conference*, pages 265–276, 1994.
15. M. Lee, M. Kitsuregawa, B. Ooi, K. Tan, and A. Mondal. Towards Self-Tuning Data Placement in Parallel Database Systems. In *Proc. of ACM SIGMOD 2000*, pages 225–236, 2000.
16. W. Litwin, M.A. Neimat, and D. Schneider. Linear Hashing for Distributed Files. In *Proceedinsg of the 1993 SIGMOD Conference*, Washington D.C., May 1993.
17. G. Matsliach and O. Shmueli. An Efficient Method for Distributing Search Structures. In *First International Conference on Parallel and Distributed Information Systems*, pages 159–166, 1991.
18. A. Mondal, M. Kitsuregawa, B. C. Ooi, and K. L. Tan. R-tree-based Data Migration and Self-tuning Strategies in Shared-nothing Spatial Databases. In *Proceedings of ACM Geographic Information Systems*, pages 28–33. ACM Press, 2001.
19. Ousterhout, J. K. G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor. Magic: A VLSI Layout System. In *21st Design Automation Conference*, pages 152–159, June 1984.
20. E. Panagos and A. Biliris. Synchronization and Recovery in a Client-Server Storage System. *The VLDB Journal*, 6(3):209–223, 1997.
21. J. Patel, J.-B. Yu, N. Kabra, and K. Tufte. Building a Scaleable Geo-Spatial DBMS: Technology, Implementation, and Evaluation. In *Proc. of the ACM SIGMOD*, pages 336–347, 1997.
22. K. Porkaew, I. Lazaridis, and S. Mehrotra. Querying Mobile Objects in Spatio-Temporal Databases. In *Proc. of 7th SSTD*, July 2001.
23. M. Prado, L. Roa, J. Reina-Tosina, A. Palma, and J.A. Milan. Virtual Center for Renal Support: Technological Approach to Patient Physiological Image. *IEEE Transactions on Biomedical Engineering*, 49(12):1420 –1430, December 2002.
24. J. Reina-Tosina, L.M. Roa, J. Caceres, and T. Gomez-Cia. New Approaches Toward the Fully Digital Integrated Management of a Burn Unit. *IEEE Transactions on Biomedical Engineering*, 49(12):1470– 1476, December 2002.
25. S. Saltenis, C. Jensen, S. Leutenegger, and Mario A. Lopez. Indexing the Positions of Continuously Moving Objects. In *Proc. of the ACM SIGMOD*, pages 331–342, May 2000.
26. B. Salzberg and V.J. Tsotras. Comparison of Access Methods for Time-Evolving Data. *ACM Computing Surveys*, 31(2):158–221, 1999.
27. P. Scheuermann, G. Weikum, and P. Zabback. Data Partitioning and Load Balancing in Parallel Disk Systems. *VLDB Journal*, 7(1), 1998.
28. B. Schnitzer and S. Leutenegger. Master-Client R-Trees: A New Parallel R-Tree Architecture. In *Statistical and Scientific Database Management*, pages 68–77, 1999.
29. A. Szalay, J. Gray, and J. van den Berg. Petabyte Scale Data Mining: Dream or Reality. In *Proc. of SIPE Astronmy Telescopes and Instruments*, August 2002.
30. T.L. Zeiler. LANDSAT Program Report 2002. Technical report, U.S. Geological Survey - U.S. Department of Interior, Sioux Falls, SD, 2002. EROS Data Center.
31. C. Zou and B. Salzberg. Safely and Efficiently Updating References During On-line Reorganization. In *Proc. of VLDB*, pages 512–522, 1998.