Effective Load-balancing via Migration and Replication in Spatial Grids

Anirban Mondal Kazuo Goda Masaru Kitsuregawa

Institute of Industrial Science
University of Tokyo, Japan
{anirban,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract. The unprecedented growth as well as the growing importance of available spatial data at geographically distributed locations has made efficient networking of such data a necessity for availability reasons. The emergence of *grid computing* coupled with large and powerful computer networks, which have the capability to connect thousands of geographically distributed computers worldwide, has opened a world of opportunities for such networking. This provides a strong motivation for designing a *spatial grid* which supports fast data retrieval and allows its users to *transparently* access data of *any* location from *anywhere*. However, several challenging issues need to be addressed for the spatial grid to work efficiently in practice. In particular, mechanisms for efficient search and effective load-balancing need to be in place. This paper focusses on dynamic load-balancing in spatial grids via data *migration/replication* to prevent degradation in system performance owing to severe load imbalance among the nodes. Notably, issues concerning load-balancing are more complex in case of grids than for traditional domains primarily because a grid usually spans across multiple administrative domains. The main contributions of our proposal are as follows. First, we view a spatial grid as comprising several clusters where each cluster is a local area network (LAN) and propose a novel *inter-cluster load-balancing* algorithm which uses migration/replication of data. Second, we present a novel scalable technique for dynamic data placement that not only improves data availability but also minimizes disruptions and downtime to the system. Our performance study demonstrates the effectiveness of our proposed approach in correcting workload skews, thereby facilitating improvement in system performance. To our knowledge, this work is one of the earliest attempts at addressing load-balancing via both *online* data migration and replication in grid environments.

1 Introduction

Spatial data occurs in several important and diverse applications associated with resource management, development planning, emergency planning and scientific research. Analysis of spatial data facilitates risk management for insurance companies, helps real estate managers to locate vacancies, enables policemen to understand crime patterns based on regions, makes it feasible for corporate organizations to develop insightful marketing strategies, capacitates effective traffic planning and allows for better understanding of the Earth system as well as its processes. Given the unprecedented growth as well as the growing importance of available spatial data at geographically distributed locations and the tremendous increase in globalization, the need for efficient networking of such data with a view towards increased data availability has never been greater. Interestingly, the emergence of *grid computing*[6] coupled with large and powerful computer networks, which have the capability to connect thousands of geographically distributed computers worldwide, has opened a world of opportunities for such networking. Grid computing relates to the massive integration and virtualization of geographically distributed computing resources, thereby enabling a grid user to see a unified image of a single *powerful* virtual computer. Notably, grid computing has been becoming increasingly important in recent years as a means of providing a feasible alternative to traditional supercomputing environments.

Scientific applications that require virtual collaboration across the globe would also benefit tremendously by deploying grids. For example, if Earth scientists could have better access to data at geographical locations that are thousands of kilometres away from their current location, their ability to compare and contrast data from distant locations would facilitate optimal exploitation of the Earth's resources. Given the current state-of-the-art, if a scientist Mr. X at Finland is interested in spatial information concerning Tokyo, he would have to first establish contact with his counterpart in Tokyo, after which he would probably be able to access the data. However, we believe it would have been much better if Mr. X could *just* log in to a system and access the relevant data concerning Tokyo. This provides a strong motivation for designing a *spatial grid* which provides acceptable response times for user queries and allows its users to access data of *any* location from *anywhere* without having to bother about the details of the procedures that mediate his access to the data.

However, several challenging issues need to be addressed for the spatial grid to work efficiently in practice. In particular, mechanisms for efficient search and effective load-balancing need to be in place. This paper focusses on dynamic load-balancing in spatial grids via data *migration/replication* to prevent degradation in system performance owing to severe load imbalance among the nodes. *Online* load-balancing strategies are preferable for availability reasons. Incidentally, issues concerning load-balancing are more complex in case of grids than for traditional domains primarily because a grid usually spans across several administrative domains. The implication is that the data at each administrative domain are likely to be managed by different administrators, thereby possibly signifying different administrative policies for indexing, load-balancing and detection of hotspots at different domains, thus exacerbating the problems associated with load-balancing.

The main contributions of our proposal are as follows.

- We view a spatial grid as comprising several clusters where each cluster is a local area network (LAN) and propose a novel inter-cluster load-balancing algorithm which uses migration/replication of data.
- We present a novel scalable technique for dynamic data placement that not only improves data availability but also minimizes disruptions and downtime to the system.

Our performance study demonstrates the effectiveness of our proposed approach in correcting workload skews, thereby facilitating improvement in system performance. To our knowledge, this work is one of the earliest attempts at addressing load-balancing via both *online* data migration and replication in grid environments. The remainder of this paper is organized as follows. Section 2 provides a brief overview of related work, while Section 3 presents an overview of our proposed system framework. The search mechanism is briefly discussed in Section 4 and issues concerning data movement are presented in Section 5. The proposed load-balancing strategy is presented in Section 6, while Section 7 reports our performance evaluation. Finally, we conclude in Section 8.

2 Related Work

Representative examples of important ongoing grid computing projects include the Earth Systems Grid (ESG)[7], the NASA Information Power Grid (IPG)[10], the Grid Physics Network (GriPhyN)[14] and the European DataGrid[3]. While the ESG project aims at facilitating detailed analysis of huge amounts of climate data by a geographically distributed community via high bandwidth networks, the IPG project attempts to improve existing systems in NASA for solving complex scientific problems efficiently. The GriPhyN project and the European DataGrid project both aim at employing grid systems for improving scientific research which require efficient distributed handling of data in the petabyte range.

Existing works [2, 18] have noted the demanding I/O needs of grid applications. While the proposal in [2] discusses the design of a data grid for data-intensive petabyte applications, the work in [18] proposes the binding of execution and storage sites together into I/O communities that participate in the wide area system. The proposal in [17] describes a data-movement system (Kangaroo) which makes opportunistic use of resources (disks and networks), while hiding network storage devices behind memory and disk buffers such that background processes handle data movements. It aims at availability and reliability by sacrificing consistency guarantees.

Static load-balancing approaches [1, 11] typically attempt to perform an intelligent initial declustering of data. Incidentally, the tile technique [13] is a commonly used declustering method for performing static load-balancing of spatial data. It works as follows. Assuming that there are P nodes in the system, the universe is first decomposed into P partitions and each of these P partitions is assigned to a different node. Then the universe is divided into T rectangular tiles of equal size such that $T \geq P$ and each tile is mapped to a partition using a hash function. Note that disjoint regions may be assigned to the same node.

However, dynamically changing user access patterns in grids implies that *no* single *static* initial data placement can guarantee good load-balancing for different kinds of access patterns, thereby necessitating dynamic approaches. Several dynamic load-balancing techniques [12, 16, 19], which adaptively balance the system load across the nodes during runtime, have been proposed. However, *none* of these works are adequate for load-balancing in heterogeneous grid environments since they do *not* take into account grid-specific issues such as heterogeneity. A notable exception to these works is the Condor system [5] which uses job (process) migration for load-balancing purposes via a 'flocking' mechanism in which multiple Condor clusters worldwide collaborate in load-sharing activities. However, process migration necessitates overheads (e.g., saving the status of a process) and can be expected to be extremely challenging, especially when the movement happens to be across different operating systems, which is often the case for grid environments. Note that our strategy differs from that of Condor since we load-balance via data migration/replication as opposed to job (process) migration.

3 System Overview

This section discusses an overview of the proposed system. In the interest of amenability, we envisage the spatial grid as comprising several clusters, where each cluster is a LAN. (Nodes are assigned to clusters such that the clusters are mutually disjoint.) This facilitates the separation of concerns between intra-cluster and inter-cluster load-balancing issues. Since intra-cluster load-balancing [12, 19] has been well researched in the traditional domain, limitations in existing works (from the grid perspective) are mostly in the realm of inter-cluster load-balancing. Hence, we shall specifically focus on inter-cluster load-balancing.

At the very outset, we define *distance* between two clusters as the communication time τ between the cluster leaders and if τ is less than a pre-specified threshold, the clusters are regarded as *neighbours.* (Since cluster leaders will collaborate, their communication time is critical.) Also, we define a node P_i as *relevant* to a query Q if it contains at least a non-empty subset of the answers to Q, otherwise P_i is regarded as *irrelevant* w.r.t. Q. Additionally, we define a cluster as being *active* with respect to Q if *at least* one of its members is still processing Q. Moreover, we shall subsequently refer to migration/replication of data collectively as *data movement* and the migrated/replicated data shall be designated as *moved data.* Notably, unlike replication, migration implies that once hot data have been transferred to a destination node, they will be **deleted** at the source node.

Most existing works define a node's load as the number of requests directed to that node, the implicit assumption being that *all* requests are of equal size, but this does *not* always hold good in practice. To take varying request sizes as well as the variations in processing capacity of different nodes into account, we define the load of node P_i, L_{P_i}, as follows.

$$L_{P_i} = D_i \times (CPU_{P_i} \div CPU_{Total}) \tag{1}$$

Here, D_i represents the number of disk I/Os at node P_i during a given time interval T_i, CPU_{P_i} denotes the CPU power of P_i and CPU_{Total} stands for the total CPU power of the cluster in which P_i is located. Also, we define the load of a cluster $L_{Cluster}$ as $\sum L_{P_i}$ i.e, the sum of the loads of its individual members. Our definition of load is in consonance with the inherent heterogeneity of grid environments where nodes are likely to be typically heterogeneous in terms of processing power and available disk space.

In our proposed strategy, each node in the grid is assigned a unique identifer *node_id* and every incoming query is assigned a unique identifier *Query_id* by the node P_i at which it arrives. *Query_id* consists of *node_id* and *num* (a distinct integer generated by P_i). Every node keeps track of the *Query_id*s that it has recently processed in order to ascertain whether it has already processed a specific query before. Additionally, every node maintains its own access statistics i.e., the number of disk accesses made for each of its data regions *only* during the *recent* time intervals for hotspot detection purposes. (Time is divided into equal pre-defined intervals at design time.) Notably, we use only recent access statistics to detect hotspots. We leave issues concerning the optimal granularity at which statistics concerning data regions should be maintained to further study.

Each cluster is randomly assigned a leader. The job of the cluster leaders is to coordinate the activities (e.g., load-balancing, searching) of the nodes in their clusters. Each node keeps track of the regions that it indexes and we shall refer to such information concerning data regions as *region-based information.* A node may store data from multiple and disjoint[1] regions in space. Additionally, each cluster leader maintains region-based information concerning the data stored both in its own cluster as well as in its neighbouring clusters. This facilitates effective pruning of the search space as it enables a cluster leader to decide quickly whether its cluster members contain the answer to a particular user query. Updates to region-based information are periodically exchanged between neighbouring cluster leaders preferably via piggybacking onto other messages.

4 Search mechanism for spatial grids

The sheer size of a grid and the huge volumes of data that it typically stores make scalability a major issue from the grid perspective. A moment's thought indicates that a centralized approach to searching *cannot* be expected to be scalable enough for grid environments. Hence, we propose a scalable distributed approach to searching in grids.

Whenever a query Q arrives at a node P_i, P_i becomes the initiator of Q. Moreover, P_i keeps track of the leaders of those clusters that are *active* with respect to Q. If P_i is *not* relevant to Q, it sends Q to its cluster leader C_i. If C_i determines from its region-based information that Q is *not* relevant to any of its cluster members, it forwards Q to members of set ξ. Set ξ comprises *only* those neighbouring cluster leaders of C_i whose members store *at least* one of the regions associated with Q. In case *none* of the neighbouring clusters have any of Q's regions, set ξ will consist of *all* the neighbouring cluster leaders of C_i. These cluster leaders, in turn, will try to answer Q via their cluster members and if their cluster members are *not* relevant to Q, the same process continues.

Once the initiator node P_i of a query Q has received the results of Q, P_i sends a message concerning termination of Q to those cluster leaders that are *active* with respect to Q and the *active* cluster leaders, in turn, broadcast the message in their respective clusters to ensure the termination of Q. However, in case of communication link failure, the termination message may *not* reach some nodes which will continue the processing of a query needlessly. To prevent such wastage of computing power, we adopt a *time-out* mechanism such that any query executing at a specific node for more than T_{max}[2] units of time should be timed-out. Interestingly, this also helps to a certain extent in preventing malicious users from degrading system performance by sending out arbitrary queries.

[1] This is similar to existing work [12].
[2] The value of T_{max} is application-dependent and is decided at design time.

5 Data movement in grids

This section discusses issues concerning data movement in grids. Since intra-cluster data movements have been well researched, any existing approach [12, 19] may be adopted. For our purposes, we adopt the approach that we had proposed in [12]. Hence, in this paper, we deal primarily with inter-cluster data movements in grids.

Migration vs Replication

Now let us study the trade-offs between migration and replication. If replication is used, in spite of the existence of several replicas of a specific data item D_i, a specific replica may keep getting accessed a disproportionately large number of times because the search is completely decentralized, thereby providing no absolute guarantee of load-balancing[3]. However, replication increases data availability albeit at the cost of disk space. In contrast, migration ensures reasonable amount of load-balancing and prevents wastage in disk space albeit at the cost of possible decreased data availability since the node to which data have been migrated may encounter failure (e.g., communication link failure, machine failure). For predicting the availability of a given node, every cluster leader monitors its nodes' availability over a period of time. If the probability of failure of a node P_i is very low, hot data should be migrated to P_i, otherwise hot data should be replicated for availability reasons. In essence, we propose that decisions concerning migration/replication should be taken during **run-time** since both migration and replication have their own inherent advantages and disadvantages.

Dealing with heterogeneity of clusters

Clusters in grid environments are highly likely to be heterogeneous in terms of processing power[4], available disk space, administrative policies (e.g., security, data access) and data management techniques (e.g., indexing, hotspot detection, load-balancing) since grids usually span across several administrative domains. Heterogeneity across clusters has significant implications for inter-cluster data movements.

Variations in indexing mechanisms: Spatial data being typically huge, it is almost always true that indexing mechanisms are used to facilitate speedy retrieval of such data. When moving data across clusters, the indexes associated with the data also need to be moved. However, variations in indexing mechanisms across clusters precludes the possibility of migrating the indexes across clusters. For example, when data are moved from a node X (which uses an R-tree [8] for indexing) to a node Y (where indexing is performed by a UB-tree [15]), the indexes of the relevant data cannot be moved *directly* from X to Y. Moreover, it may *not* be possible to integrate the moved data smoothly into Y's index structure.

We address this problem by extracting data from the index at the source node and transferring the data to the destination node. At the destination node, there are two different indexes, one for organizing the dedicated data at that node and another for the moved data. Note that at the destination node, the moved data are indexed by the indexing mechanism of the destination node itself, irrespective of the indexing mechanism at the source node of the data. This is important because the destination node may *not* have the indexing software that was being used to index the data at its source node. Interestingly, moving data as opposed to moving indexes also solves problems associated with *porting*. In retrospect, it is clear that in practice, we *cannot* impose our grid-related policies on any cluster, thereby indicating that our policies should be aimed at *supplementing* a given cluster's policies as opposed to interfering with them.

Variations in available disk space: Given the implications of significant variations in available disk space of different nodes in a grid, we adopt the following strategy.

[3] If the same query is issued from different nodes, randomness may guarantee a certain amount of load-balancing.

[4] Just to recapitulate, our definition of load takes into consideration variations in processing power.

- 'Pushing' non-hot data (via migration for large-sized data and via replication for small-sized data) to large capacity nodes as much as possible.
- Replicating small-sized hot data at small capacity nodes (smaller search space expedites search operations).
- Large-sized hot data are migrated to large capacity nodes only if such nodes have low probability of failure, otherwise they are replicated at large capacity nodes, an upperlimit being placed on the number of replicas to save disk space.

Consistency issues

Interestingly, the spatial attributes of spatial data are relatively static, while updates on the non-spatial attributes are possible. For example, the spatial coordinates of a hotel should remain the same, but the number of available rooms (a non-spatial attribute) may be updated. In such cases, *strict consistency* is *not* absolutely critical. Our data movement technique sacrifices both replica consistency as well as disk-cache coherence to provide improved data availability. Notably, this is in consonance with prominent grid data efforts [2, 9, 17] which have noted the static nature of many scientific datasets.

In our lazy replica consistency maintenance scheme, every data item δ_i has a single *primary copy* at node P_i and possibly some replicas. Each node keeps track of migrated/replicated data at itself and also maintains information concerning the primary source[5] and the immediate source of the migrated/replicated data. Any update to δ_i can *only* be performed on the primary copy and then the update is propagated to other nodes in a lazy manner via piggybacking techniques. In case a user desires to access fresh data, his/her query will be directed to the node which stores the primary copy of the requested data. Additionally, each node periodically checks the number of accesses N_k (during the recent time intervals) for the *moved* data that it stores to ascertain which items are still hot and the *moved* items, for which N_k falls below a pre-specified threshold, are deleted unless a given *moved* item is a primary copy. Such periodic 'cleanup' of *moved* data results in more available disk space and is important from the perspective of system scalability over time.

Suppose a data item δ_i has been moved from a node P_i to a node P_j. If a copy of δ_i exists in P_i's cache, it is allowed to remain in P_i's cache. Queries for δ_i are answered from P_i's cache even though δ_i does *not* exist in P_i's disk. Notably, this is very different from work in traditional domains. When δ_i becomes cold, it will automatically get discarded by the caching scheme. If subsequently, δ_i is updated at P_j, P_j informs P_i via a piggy-backed message concerning the update and the cached copy of δ_i at P_i is flushed. Since P_j may *not* inform P_i immediately about the update, some queries accessing the cached copy of δ_i at P_i may read obsolete data. Thus, we sacrifice disk-cache coherence to some extent to provide faster response times to the users.

6 Load-balancing

This section discusses load-balancing via migration and replication of data from heavily loaded nodes to lightly loaded nodes. While intra-cluster load-balancing [12, 19] refers to balancing the loads within a particular cluster, inter-cluster load-balancing attempts to ensure load-balancing among the clusters i.e., to achieve load-balancing across the system as a whole.

Intra-cluster load-balancing

Intra-cluster load-balancing has been well researched in the traditional domain, but for grid systems, we should also take into account varying available disk capacities of nodes within a

[5] Primary source refers to the node which contains the primary copy of the data.

cluster. Our strategy is as follows. The cluster leader (say C_i) periodically receives information concerning loads L_i and available disk space D_i of the nodes and initially creates a list $List$ by sorting the nodes (in descending order) based *only* on L_i such that the first element of $List$ is the most heavily loaded node. Assume there are n elements in $List$. Among the last $\lceil n/2 \rceil$ nodes in $List$, the nodes whose respective values of D_i are less than a pre-specified threshold are deleted from $List$. Then load-balancing is performed by migrating or replicating hot data from the first node in $List$ to the last node, the second node to the second-last node and so on. Data are only moved if the load difference between the nodes under consideration exceed a pre-specified threshold. Note that this is similar to our previous work [12] except that in this case, we also take variations in available disk space into consideration. For decisions concerning the initiation of data movement, the amount of data to move, determination of the data to move (based on hotspot detection mechanism), identification of the source and destination nodes, we adopt the same approach as in [12].

Inter-cluster load-balancing

To prevent load imbalance among clusters, inter-cluster load-balancing becomes a necessity. We propose that such load-balancing should be performed *only* between neighbouring clusters by means of collaboration between the cluster leaders, the reason being that moving data to distant clusters may incur too high a communication overhead to justify the movement.

Cluster leaders periodically exchange load information *only* with their neighbouring cluster leaders. If a cluster leader α detects that its load exceeds the average loads of the set β of its neighbouring cluster leaders by more than 10% of the average load, it first ascertains the hot data regions that should be moved and sends a message concerning each hot region's space requirement to each cluster leader in β in order to offload some part of its load to them. The leaders in β check the available disk space in each of their cluster members and if their disk space constraint is satisfied, they send a message to α informing it about their total loads and their total available disk space. α sorts the willing leaders of β in $List_1$ such that the first element of $List_1$ is the least loaded leader.

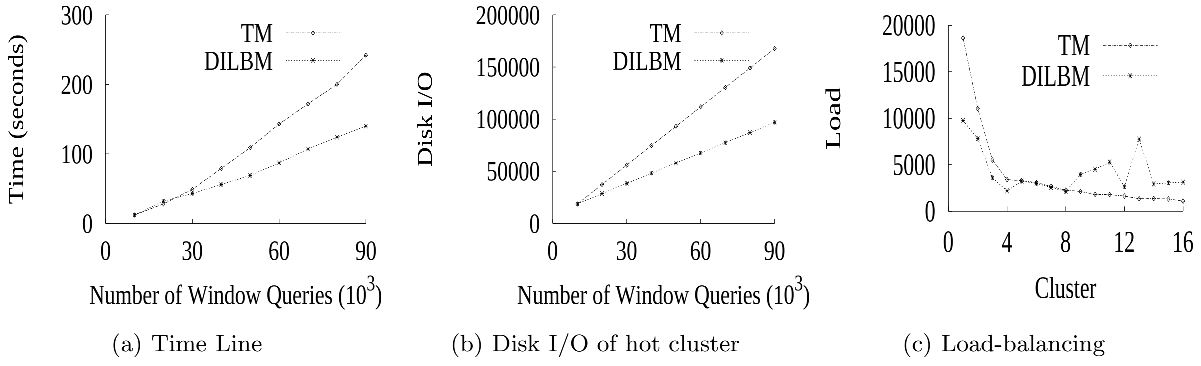
The number of hot data regions to be moved depends upon load imbalance i.e, if the load imbalance is severe, more data regions need to be moved, while in case of lesser load imbalance, lesser data regions are moved. Assume the hot data regions to be moved are numbered as h_1, h_2, h_3,h_4... (h_1 is the hottest element). Let the number of willing nodes in β and the number of hot data regions to be moved be denoted by b and h respectively. If $b < h$, h_1 is assigned to the first element in $List_1$, h_2 is assigned to the second element and so on in a round-robin fashion till all the hot regions have been assigned. If $b \geq h$, the assignment of hot data to elements of $List_1$ is done similarly, but in this case some elements of $List_1$ will *not* receive any hot data. After the hot data arrives at the destination cluster's leader, the leader creates a sorted list $List_2$ (in ascending order according to load) of its nodes and assigns the hot data to elements of $List_2$ in the manner described above.

7 Performance Study

This section reports the performance evaluation of our proposed inter-cluster load-balancing technique via data migration[6]. In the near future, we also intend to study the impact of replication since we expect that replication can improve system performance further. Note that we consider performance issues associated *only* with inter-cluster load-balancing since a significant body of research work pertaining to efficient intra-cluster load-balancing algorithms already exists. We specifically study the performance of our proposed scheme with variations in the workload skew.

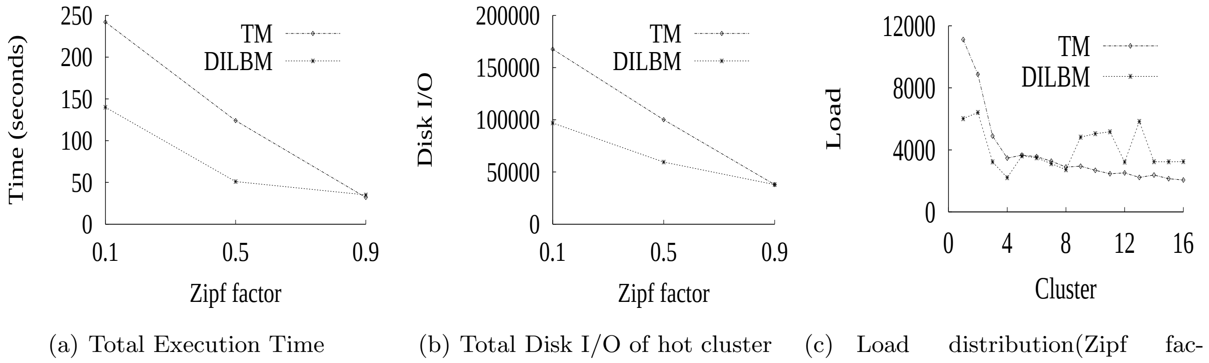
[6] Owing to constraints concerning available disk space, we could not investigate replication in our performance study.

Our test environment comprises a cluster of 16 SUN workstations, each of which is a 143 MHz Sun UltraSparc I processor (256 MB RAM) running Solaris 2.5.1 operating system. These are connected by high speed switch (200 Mbyte/s), the APnet. Each cluster is modeled by a workstation node in our experiments. Hence, throughout this section, we shall be using the term 'cluster' to refer to such a workstation node. From this perspective, such a workstation node may be viewed as the cluster leader which is representative of its entire cluster. The implication is that there are 16 neighbouring clusters among which we attempt to provide inter-cluster load-balancing. Additionally, to model inter-cluster communication in a wide area network environment, we simulated a transfer rate of 1 MB/second among the respective clusters.



(a) Time Line (b) Disk I/O of hot cluster (c) Load-balancing

Fig. 1. Performance of our proposed scheme



(a) Total Execution Time (b) Total Disk I/O of hot cluster (c) Load distribution(Zipf factor=0.5)

Fig. 2. Effect of varying skew in the queries

We implemented an R-tree on each of the clusters to organize the data allocated to each cluster. For all our experiments, we assumed that one R-tree node fits in a disk page (page size = 4096 bytes). Hence, R-tree node capacity is the same as page size in our case. The height of each of the R-trees at each node was 3 and the fan-out was 64. We have used the tile technique [13] as reference[7]. We divided the universal space into 48 tiles and the tiles were assigned to the 16 clusters. We shall henceforth refer to this approach as **TM** (tile method). Also, we shall refer to our proposed strategy as **DILBM** (dynamic inter-cluster load-balancing via migration). A real dataset (Greece Roads [4]) was enlarged and used for our experiments. The enlargement was done by translating and mapping the data. Each cluster contained more than 200000 spatial data rectangles. In order to model skewed workloads, we have used the well-known Zipf distribution to decide the number of queries to be directed to each cluster. Note that this is only an approximate manner of generating skewed workloads since queries may vary in selectivity.

7.1 Performance of our proposed scheme

Now let us investigate the effectiveness of our proposed scheme in improving the system performance. For this purpose, an experiment was performed using 90000 spatial select (window) queries, the value of the zipf factor being set at 0.1, which indicates a highly skewed workload.

Figure 1a displays a time line indicating the progress of the queries over time by presenting the wall-clock execution times of queries as a function of the time intervals during which the queries were executed, while Figure 1b presents the disk I/O of the hot cluster for the same experiment. Figure 1a indicates that initially during certain intervals, the performance of DILBM is slightly worse than that of TM. This occurs owing to migration-related overheads and disturbances. However, once the migration has been completed, DILBM significantly outperforms TM. This is possible because of the reduction in the load of the hot cluster as demonstrated in Figure 1b. Such reduction occurs as a result of the effective load-balancing provided by DILBM as depicted in Figure 1c, which indicates that DILBM is capable of distributing the load more evenly than TM (especially reducing the load of the hot cluster, namely cluster 1).

7.2 Variations in Workload Skew

Now we shall examine the performance of DILBM for varying skews in the workload. For this purpose, we used zipf factors of 0.5 and 0.9 to model medium-skewed workload and low-skewed workload respectively.

Figure 2a depicts the wall-clock completion time of *all* the queries when the zipf factor was varied, while Figure 2b shows the total disk I/Os incurred by the hot cluster during the same experiment. Interestingly, the gain in execution time is significantly more in the case of highly skewed workloads. This gain keeps diminishing as the query skew diminishes, till at some point, there is *no* significant gain at all. This occurs because as the workload skew decreases, the need for load-balancing also reduces owing to the query pattern itself contributing to load-balancing. Note that at a value of 0.9 for the zipf factor, there is no significant difference in performance between DILBM and TM since the workload in this case was too lowly skewed to necessitate migrations. Incidentally, DILBM performs slightly worse than TM for lowly skewed workloads primarily owing to overheads incurred in making the decision that the skew is too low to necessitate migrations. However, we believe this is a small price to pay as compared to the big gain achieved by DILBM in the case of highly skewed workloads. Figure 2c demonstrates that DILBM performs reasonably well for medium-skewed workloads, especially in reducing the load of the hot cluster. In essence, DILBM is sensitive to changing workload distributions and adapts very well indeed.

8 Conclusion

Huge amounts of available spatial data at geographically distributed locations coupled with the growth of powerful computer networks provides a strong motivation for designing a spatial grid.

[7] Just to recapitulate, the tile technique is a commonly used technique for declustering spatial data and supports static load-balancing.

Efficient search and load-balancing mechanisms are essential for the grid to work efficiently in practice. Incidentally, issues concerning indexing and load-balancing are more complex in case of grids than in case of traditional domains primarily because a grid usually spans across several administrative domains. We have proposed a dynamic strategy for inter-cluster load-balancing in spatial grids via data migration/replication and analyzed the trade-offs between migration and replication. Our performance study has demonstrated that our proposed technique is indeed feasible for providing load-balancing in spatial grids. Currently, we are investigating performance issues concerning replication in spatial grids.

References

1. H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez. Prototyping Bubba, a highly parallel database system. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), March 1990.
2. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Proc. Network Storage Symposium*, 1999.
3. European DataGrid. http://eu-datagrid.web.cern.ch/eu-datagrid/.
4. Datasets. http://dias.cti.gr/∼ytheod/research/datasets/spatial.html.
5. D. H. J Epema, M. Livny, R. V. Dantzig, X. Evers, and J. Pruyne. A worldwide flock of Condors : Load sharing among workstation clusters. *Journal on Future Generations of Computer Systems*, 12, 1996.
6. I. Foster and C. Kesselman. The grid: Blueprint for a new computing infraestructure. *Morgan-Kaufmann*, 1999.
7. Earth Systems Grid. http://www.earthsystemgrid.org/.
8. A. Guttman. R-trees: A dynamic index structure for spatial searching. *In Proc. ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.
9. W. Hoscheck, J. Jaen-Martinex, A. Samar, H. Stockinger, and K. Stockinger. Data management in an international data grid project. *Proc. IEEE/ACM International Workshop on Grid Computing*, 2000.
10. NASA IPG. http://www.ipg.nasa.gov/.
11. N. Koudas, C. Faloutsos, and I. Kamel. Declustering spatial databases on a multi-computer architecture. *In Proc. EDBT*, pages 592–614, 1996.
12. 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. *Proc. ACM GIS*, 2001.
13. J. Patel and D. DeWitt. Partition based spatial-merge join. *In Proc. ACM SIGMOD International Conference on Management of Data*, pages 259–270, 1996.
14. GriPhyN Project. http://www.griphyn.org/index.php.
15. F. Ramsak, V. Markl, R. Fenk, M. Zirkel, K. Elhardt, and R. Bayer. Integrating the UB-tree into a database system kernel. *In Proc. VLDB International Conference on Very Large Databases*, pages 263–272, 2000.
16. P. Scheuermann, G. Weikum, and P. Zabback. Adaptive load balancing in disk arrays. *Proc. Foundations of Data Organization and Algorithm*, pages 345–360, 1993.
17. D. Thain, J. Basney, S.C. Son, and M. Livny. The Kangaroo approach to data movement on the grid. *Proc. HPDC*, 2001.
18. D. Thain, J. Bent, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Gathering at the well: Creating communities for grid I/O. *Proc. SC*, 2001.
19. Gerhard Weikum, Peter Zabback, and Peter Scheuermann. Dynamic file allocation in disk arrays. *In Proc. ACM SIGMOD*, pages 406–415, 1991.