Dynamic Data Migration Policies for Query-Intensive Distributed Data Environments*

Tengjiao Wang[1], Bishan Yang[1], Allen Huang[2], Qi Zhang[1], Jun Gao[1], Dongqing Yang[1], Shiwei Tang[1], and Jinzhong Niu[3]

[1] Key Laboratory of High Confidence Software Technologies, Peking University, Ministry of Education, China
School of Electronics Engineering and Computer Science, Peking University
Beijing, 100871 China
{tjwang, bishan_yang, rainer, dqyang, gaojun, swtang}@pku.edu.cn
[2] Microsoft SQL China R&D Center
allen.huang@microsoft.com
[3] Computer Science, Graduate School and University Center, City University of New York
365, 5th Avenue, New York, NY 10016, USA
jniu@gc.cuny.edu

Abstract. Modern large distributed applications, such as telecommunication and banking services, need to respond instantly to a huge number of queries within a short period of time. The data-intensive, query-intensive nature makes it necessary to build these applications in a distributed data environment that involves a number of data servers sharing service load. How data is distributed among the servers has a crucial impact on the system response time. This paper introduces two policies that dynamically migrate data in such an environment as the pattern of queries on data changes, and achieve query load balance. One policy is based on a central controller that periodically collects the query load information on all data servers and regulates data migration across the whole system. The other policy lets individual server dynamically selects a partner to migrate data and balance query load in between. Experimental results show that both policies significantly improve system performance in terms of average query response time and fairness, and communication overhead incurred is marginal.

1 Introduction

As more and more query-intensive applications, such as telecommunication and banking services, are running in large distributed data environments, it is a

* This work is supported by the Cultivation Fund of the Key Scientific and Technical Innovation ProjectMinistry of Education of China(No.708001), the National '863' High-Tech Program of China(No.2007AA01Z191,2006AA01Z230), and the NSFC(Grants 60873062).

Q. Li et al. (Eds.): APWeb/WAIM 2009, LNCS 5446, pp. 63–75, 2009.

major concern for these service providers how to serve millions of requests with short response time. These environments typically involve a number of data servers sharing service load and the distribution of data among these servers has a crucial impact on how fast queries are fulfilled in average.

A typical example of these applications is telephony service. When a user dials a number to make a call, a request is made to query the callee's location information so as to establish a connection between the two parties. Typically each user's location information is stored as a record in a database that spans across a set of distributed database servers, as the overall data volume is massive. Assuming that each record is unique system-wide, a query on a particular record needs to be routed to the server on which the record is stored. It is usually the case that the number of queries on records are not evenly distributed. A high access frequency of a small set of data records on a single server may easily overload the server; while some other servers may be basically idle if their data is rarely accessed. When load imbalance is severe, the average query response time may easily surpass the maximal time human users can stand. Therefore, how the records are distributed across the system is an important issue for system designers.

Generally, these systems share the following features:

- A massive amount of data is distributed among multiple database server nodes without duplicates. That is a data record is available on exactly one node in the system and queries on the data have to be routed to the node.
- Fast response and high efficiency in serving queries are guaranteed even when the overall query load is high system-wide.
- Data distribution has a significant impact on query load distribution. When data is unevenly distributed among nodes, load imbalance occurs and queries on some records may have to wait a long time to get served.
- The pattern of queries may change over time, and as a result, data records need to be migrated between server nodes dynamically to balance workload.

1.1 Related Work

Load balancing involving data distribution has been addressed in [1,2]. Their approaches are however used to allocate web documents among a cluster of web servers in order to achieve load balance, which is different from the problem we would like to tackle.

A relating problem domain is data distribution in parallel storage systems. In these systems, *horizontal data partitioning* has been commonly used to distribute data among system nodes. There are further three different strategies for horizontal partitioning: *round-robin*, *hash*, and *value-range partitioning*. The value-range partitioning may create skewed data distributions. The round-robin partitioning is capable of allocating data approximately evenly between partitions, but it requires brute-force searches and is ineffective for queries. The hash partitioning can not only obtain even data distribution, but also perform well for exact matching queries. Although these partitioning strategies each have both

advantages and disadvantages, they do not take into consideration the access pattern on data, which would be a problem when the pattern is not a uniform distribution. Dynamic data reallocation is necessary to effectively adapt for changes of access patterns. When access frequencies of data items lead to unbalanced workload on system nodes, data migration should be performed to balance workload. The simple data skew handling method in [3] balances the storage space for data, but does not guarantee balanced data access load across system nodes. Similarly most access load skew handling methods do not consider data balancing [4,5]. In [6], a data-placement method was proposed to balance both access load and storage space of data, but it is applied in parallel storage system, and focuses on achieving availability and scalability for parallel storage configuration.

In this paper, we introduce a series of effective policies for dynamic data migration in query-intensive distributed data environments. Our work differs from previous work in that our approach models patterns of end-user queries and performs dynamic data migration to speed up query processing. To adapt for the ever-changing load distribution in the system, we design dynamic data migration policies to do query load balancing. The idea is basically to collect query load information at system nodes periodically and perform data migration automatically to obtain load balance in the system. In particular, our work has the following contributions:

- We propose a dynamic data migration policy with centralized control. The data migration process is controlled by the central controller that makes decisions from the system-wide viewpoint.
- We propose a dynamic data migration policy with decentralized control. Each individual node has its local viewpoint and makes decisions concurrently along others.
- We develop a platform to evaluate the effectiveness of our proposed policies. Experimental results show that these policies perform almost optimally, and are able to significantly reduce query response time even with the varying query load across the system.

The remainder of this paper is organized as follows: Section 2 presents a model for the query-intensive distributed data environments; Section 3 describes the two proposed dynamic data migration policies in details, one with centralized control and the other with distributed control; Section 4 describes the experimental environment and presents an experimental comparison between the three strategies; and Section 5 concludes this article.

2 Model Description and Formulation

This section presents a query-intensive distributed model, which is commonly used in mobile communications[7]. As Fig. 1 depicts, the model can be divided into three parts: back-end (BE) servers, front-end (FE) servers and application clients (ACs).

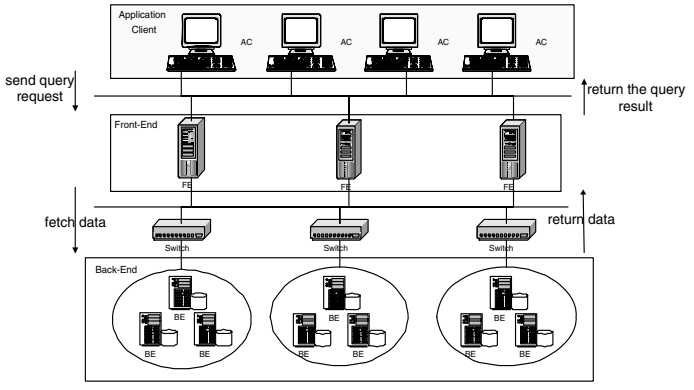

Fig. 1. A model for query-intensive distributed data environment

The BE servers hold data that would be queried about and together form a huge database system. Here we assume that data is distributed between BE servers without overlap and can be migrated between them.[1] The databases on BE servers have a homogeneous structure, but the processing powers of these BE servers may differ. When a user sends a query through an application client, it first arrives at one of the FE servers and then is routed to the BE server where the queried data is located. The BE server then executes the query on its local database, and returns the requested information.

We now make several definitions that would be needed in subsequent sections.

Definition 1. *The query load associated with a data unit is the frequency of read operation*[2] *on it in a unit of time, denote as* l.[3]

Definition 2. *Given a* data group, G, *which has* k *data units with their corresponding loads being* $l_1, l_2, ..., l_k$ *respectively, the* query load *associated with* G *is defined as* $L = \sum_{j=1}^{k} l_j$.

Definition 3. *The* process power *of a BE sever,* μ, *is the highest query frequency it can deal with within a unit of time. If there are* k *data groups on the BE server and* $L_1, L_2, ..., L_k$ *are their query loads respectively, the* load level *of the BE server is defined as* $\beta = \frac{\sum_{i=1}^{k} L_i}{\mu}$.

[1] Multiple BE servers could possibly be used to hold a same set of data units for higher reliability, with one of them being the main sever and the others being backups.

[2] We only consider read operations here because in the environment read operations are the main operation.

[3] In our experiment one unit of time is 10 seconds, which is common in most applications in our application domain.

3 Dynamic Data Migration Policies

To adapt the fluctuation of query load in the system, dynamic data migration policies are needed. We propose two dynamic data migration policies that can adapt for the changing pattern of query load, one with centralized control and the other with distributed control.

Either migration policy consists of three components: (1) information rule, which defines how the query load information is collected and maintained; (2) selection rule, which regulates the selection of source server nodes, with high load, and destination server nodes, with low load, for data migration; (3) migration rule, which determines when and how to migrate data.

3.1 DDMC: Dynamic Data Migration with Centralized Control

We now present the dynamic data migration policy with a central controller (CC). With centralized control, the global information of the query load is collected by asking each BE server to inform the central center of its local load. Data migration decisions are made by the central controller itself. The processing power of the central controller is assumed high enough to fulfill the task. It can be implemented by a low cost processor or even run as a single process on an existent node in the system.

Information Rule. Suppose the query load of data group G_i on FE_j is denoted as $L_{i,j}$, and there are totally N_{fe} FE servers in the system, the total query load of data group G_i is then the sum of query loads recorded on all the FE servers, expressed as $L_i = \sum_{j=1}^{N_{fe}} L_{i,j}$. Since the data groups have no overlap across BE servers, we can calculate the load level of BE_k as the ratio of the overall load of the data groups on BE_k and BE_k's processing power $\beta_k = \sum L_j/\mu_i$.

To avoid high communication overhead, CC periodically notifies all the FE servers to collect load information of each data group in their data group tables. The gathered information is also stored in a table on CC, which has the same structure as the FE data group tables though it records the global information of data groups. CC also maintains a table, called T_{BE}, as a whole picture of the BE servers. Each entry of T_{BE} contains information about a BE server, including its *BE's ID*, its *processing power*, and its current *load level*.

Selection Rule. The objective is to balance the query load in the system through data reallocation. Since BE servers may have various processing capabilities, we can not simply redistribute data to make an uniform load distribution. Intuitively, if all the nodes are at the same level of utilization, the resources are considered to have been made best use of and the servers have achieved system-wide balance. We classify the BE servers into three categories: HL (Heavily Loaded), NL (Normally Loaded), and LL (Lightly Loaded). The classification is based on the average load level of the system: $AvgL = (\sum_{i=1}^{N} Ł_i)/(\sum_{i=1}^{N} \mu_i)$.

The system is viewed as in an optimal state when the load levels of each BE server converged to to $AvgL$. With a toleration constant ϕ, a node is considered a HL one if its load level $\beta > AvgL + \phi$, a NL one if $AvgL - \phi \leq \beta \leq AvgL + \phi$, or a LL one if $\beta < AvgL - \phi$. All the HL nodes are chosen as source node candidates and LL nodes are chosen as destination node candidates.

The basic idea is to balance loads between HL nodes and LL nodes. First a HL node is selected, then a LL node is selected for data migration between the two parties. Due to the communication overhead in a large distributed network, only a small set of LL nodes should be selected for a particular HL node. We, in particular, choose the LL node from the k nearest neighbors to the source node. To further minimize the migration overhead, we assign each destination node candidates a *priority*, which represents the priority for it to be chosen. For a destination node candidate BE_i, the priority, p_i, is defined as: $p_i = (\frac{\mu_i * (AvgL+\phi) - L_i}{\max_j \{\mu_j * (AvgL+\phi) - L_j\}})$ The priority reflects how much potential a LL node has to take more query load. We select the node with the highest priority as the destination node.

Migration Rule. To speed up data migration, the algorithm is divided into two stages: *decision making* and *data migrating*.

First, the source node candidate with the highest load level is chosen as a source node, called BE_i, and the destination node candidate with the highest priority is chosen as a destination node, called BE_j. In order to move less data, the data groups on BE_i are considered in descending order of their corresponding load values. A data group that is eligible for migration if it satisfy the following two conditions:

Condition 1. $\omega^c \delta_i > \Delta$, where c is the number of times the data group has been migrated, ω $(0 < \omega \leq 1)$ is a weighting factor used to prevent the data group from being migrated too frequently, δ_i describes how heavier the load level of the source node is relatively, and Δ (> 0) is a constant used to protect the system from potential instability. In particular, δ_i is defined as: $\delta_i = \beta_i / \max_k \beta_k$ where $\max_k \beta_k$ is the highest load level in the system. When $\omega = 1$, c will not effect the data migration decisions, and only the load level of the node matters; and when $\omega = 0$, no data groups would be migrated.

Condition 2. After the migration, the load level of the source node should not be lower than $AvgL - \phi$ and the load level of the destination node should not be higher than $AvgL + \phi$, which can be expressed as $\beta_i - l/\mu_i \geq AvgL - \phi$ and $\beta_j + l/\mu_j \leq AvgL + \phi$, where l is the query load of the data group to be migrated from the source node to the destination node.

It is possible that there are no data groups on the source node satisfying the above two conditions. Under this case, data groups on the destination node will also be taken into consideration and we allow data exchanging between the source node and the destination node. A data group with load l on the source

node BE_i and a data group with load l' on the destination node BE_j would be exchanged if $\beta_i - (l - l')/\mu_i \geq AvgL - \phi$ and $\beta_j + (l - l')/\mu_j \leq AvgL + \phi$destination node should be searched in the order of increasing load. The first one that meets the above conditions is selected for exchanging.

Once the migration decision is made, the central controller will control the migration process. The priority of the destination node will be updated, and the new destination node will be chosen after migration. This procedure is repeated until all the data groups on the source node are explored for the migration possibilities. Then the next source node candidate is chosen. Until the iterative loop on the source node candidates is finished, the decision process is ended.

Data Migration Mechanism. We provide a data migration mechanism for ensuring data consistency during the migration process. The exchanging process can be divided into two migration processes, by exchanging source and destination.

Step 1. At the beginning, CC sends a command to source node BE_s for migrating data D to destination node BE_d.
Step 2. BE_s performs the migration.
Step 3. If BE_d successfully receives and stores the data, it sends a success signal to CC.
Step 4. CC broadcasts messages to all FE servers to ask for location update in their data group tables.
Step 5. Each FE server performs location update, and then sends a success signal to CC.
Step 6. After receiving the success signals from all FE servers, CC sends a command to BE_s for deleting the migrated data D.
Step 7. BE_s performs the deletion.

3.2 DDMD: Dynamic Data Migration with Distributed Control

This section provides an alternative approach without any centralized control. Individual BE server makes the data migration decision based on their own situation. With this policy, each BE server calculates $AvgL$ by broadcasting its own information to other BE servers in the system. Similar to the policy with centralized control, the BE servers are classified into three categories: HL, NL and LL.

The policy is a HL-initiating one. The objective is to lighten the query load in HL nodes by migrating their frequently queried data to the LL nodes. Only the HL nodes need to store information about the LL nodes. This will enable the HL nodes to make migration decisions, i.e., choosing a LL node that is the best for migration. A LL node may receive migration requests for different HL nodes at the same time. In this case, the LL node will choose the one it thinks the most deserved for migration. The decision of the HL node is valid only if the decision is accepted by the chosen LL node. In this way, a HL node that is too desperate in reducing its load is regulated to some extent.

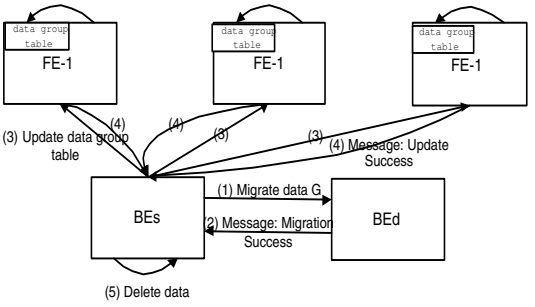

Fig. 2. Data migration process with distributed control

Information Rule. To obtain the overall load, each BE server requests all the FE servers to collect the corresponding data groups information. Then it calculates and stores its own load value L and the load level β, and classifies itself to HL, NL or LL in the light of the threshold. After that, each BE server broadcasts their own state to other BE servers in the system. In this way, every BE server can know the average load level, $AvgL$, of the system. If a server finds itself as a LL node, it will build up a HL-list to record all the HL nodes's IDs, for knowing where its information has been kept. If a server is a HL node, it will build up a LL-List and store the load values of the LL nodes from their broadcasting. The aim of this process is to build up the necessary information for migration decisions.

Selection Rule. The basic idea is that each HL node chooses the destination node candidate that most appropriate for migration regardless of other HL nodes's decisions. On the other hand, the LL node tries to let the most needed HL node to migrate data. This is a concurrent decision making process. Individual HL node can making decisions simultaneously based on their own viewpoints of the system. Specifically, they have different description about the LL nodes in the system, owing to the distance factor. The method used to measure the priority of the LL node is the same way as that used in the centralized policy. It can be computed locally since the HL node knows the current load and the processing power.

Migration Rule. Based on the selection rule, the HL node chooses the destination node based on the priority, from high to low. Since the selection process in various HL nodes can be taken place at the same time, it is likely that a LL node is selected as a destination node by more than one HL nodes. To maintain a coherent view of the system, the load copies of the LL nodes must be consistent. To address this coherency problem, the selected LL node should choose only one source node, and make other HL nodes in the HL-List to see its new load.

The migration mechanism for DDMD is introduced in the follow. It only deals with the information update for the migrated data, and the overall load value of individual node is assumed to get consistent through the above communication process. Fig. 2 shows the mechanism.

Step 1. the data group migrates from the source node to the destination node.
Step 2. if the destination node successfully receives the data, it return a success signal to the source node.
Step 3. the source node notifies all the FE servers to update the location information of the migrated data.
Step 4. All the FE servers performs the update and then sends a signal to the source node.
Step 5. When the source node receives signals from all the FE servers, it deletes the data and its corresponding information in the local table.

This mechanism can also be extended to apply in data exchanging by replacing the source node by a LL node and the destination node by a HL node.

4 Experiments

For the purpose of this study, we prepare real customer data from a mobile communication company. It contains 6,000,000 users covering attributes like the basic ID of the user, the current location of the user and IMSI (International Mobile Subscriber Identification Number). All the data is stored in database, and is queried and processed through SQL sentence. Our platform consists of six BE servers and one FE server. Each server runs a DB2 V8.0 database. The FE server is responsible for dispatching an incoming query request to one of the BE servers. The hardware configurations are presented in Table 1. The processing rate is measured by the number of queries being dealt with by 600 threads in a second, and the processing rate of the slowest computer is set to 1. All the system nodes are assumed to be connected by a communication network with 100Mbps bandwidth. A query was created by generating an id number, which was used as the key for searching the users' location information in the database. They are generated according to a Zipf distribution, the Zipf factor is varied from 0.7 to 0.9. The reason is that in real case, only a few data is queried with high frequency in a period, a medium amount of data with middle query frequency, and a large amount of data is queried not so continually.

Table 1. System configuration

CPU	Memory	Number	Process power(relative)
2.0GHz P4	512M	1	1
4*1.8GHz AMD Opteron 865	18GB	1	5
AMD Athlon 64 3200+	1G	4	2

4.1 Description of Experiments

To simulate the queries in the real applications, we generate 600 threads to send queries to the system concurrently, each thread carrying several SQL queries. We found from experiments that if the number of overall queries reach 6000, the system resources are in the largest extent of use. Hence we generate queries from 3000 to 6000 to vary the system load level from 0.5 to 1.0.

For comparison purpose, an algorithm NoLB was used. It distributes data to databases based on the commonly used horizontal fragmentation method and does not consider any query load information. A static data distribution algorithm DAH in [9] is also used for comparison.

Initially, the data is distributed by NoLB. The data group size is set to be 100 records. The system load level is defined as the ratio of total query frequency to the overall processing power of the system: $AvgL = (\sum_{i=1}^{N} Ł_i)/(\sum_{i=1}^{N} \mu_i)$. At initial state, we set the system load level at 0.5. Then we varied it from 0.5 to 1.0.

Comparisons between DAH, DDMC, and DDMD. In the second experiment, the effectiveness of the dynamic data migration policies are evaluated, including DDMC and DDMD. After the system query load is moderated by DAH, we vary the query distribution a little by tuning the Zipf factor. At this time, the previous distribution is not adaptable, and DDMC and DDMD policies are considered. As for DDMC, the central controller periodically collected the query load information from the FE server at a time interval of 10 minutes. Similarly for DDMD, BE servers gather information from each other at each 10 minutes. A large number of runs for DDMC and DDMD were conducted with different values of adjustable parameters, and the best combination of those parameter values were used. The tolerable deviation of the average query load ϕ is set at 10% of $AvgL$.

In Fig. 3, we present the average response time of DDMC, DDMD and DAH with the increasing load level of the system. From the result in Fig. 3, DDMC and DDMD both provide substantial speedup of the average response time of the

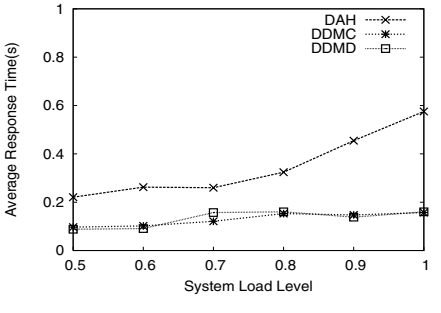

Fig. 3. Average response time

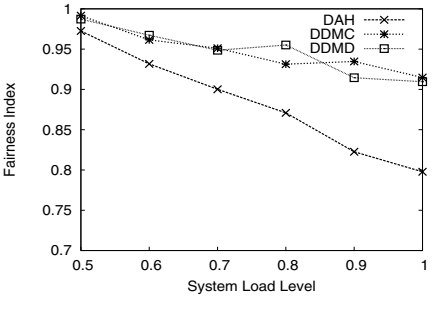

Fig. 4. Fairness indices with DAH, DDMC, and DDMD respectively

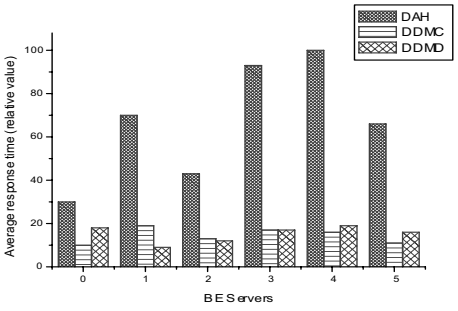

Fig. 5. Average response time at individual servers

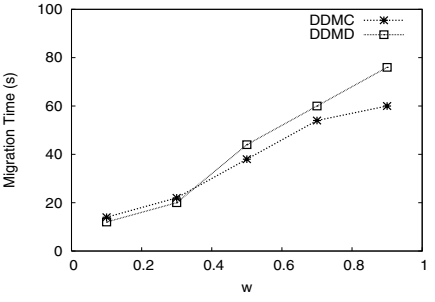

Fig. 6. Migration time with different migration weights

system over the situation where there are no dynamic migration decisions. This is because dynamically data migration can contribute to achieve load balancing and adapt the changing query patterns, and the overall performance of the system can be expected to be improved. The comparison results are more obvious with higher load level of the system. The performance provided by DDMD is close to that provided by DDMC. This confirms the fact that making decisions based on individual server's viewpoint will not be inferior to the system-wide viewpoint.

The average response time for individual BE server before and after dynamic migration is also given. Fig. 5 shows the average response time when the system's load level is at 0.7. Both the two policies moderate the response time and make much better utilization of the system resources. Each BE server's load level is inside the expected zone of the average load level.

Fairness index I proposed in [8] is used as a measure of balance: $I = \frac{[\sum_{i=1}^{n} F_i]^2}{n \sum_{i=1}^{n} F_i^2}$ where F_i is the expected response time of node i. If the expected response time is the same for all nodes, then $I = 1$, which means the system achieves fairness distribution. Otherwise, if the expected response time differs significantly from one node to another, then I decreases, which means the system suffers load imbalance to some extent. With respect to fair index, Fig. 4 shows that DDMC and DDMD has a close fairness index of about 0.91 to 1.0. The fairness index of DAH varies from 0.97 at high load to 0.79 at low load, while DDMC and DDMD shows stable fair index value with the increasing load level of the system.

Comparisons between DDMC and DDMD on Communication Overhead. In the dynamic policies, the communication overhead costs caused by gathering and exchanging information may negate the benefits of them. In Fig. 6, we show cost of the dynamic migration algorithms DDMC and DDMD in terms of their communication time.

To examine how communication overhead changes at different levels of data migration, we experimented with ω varying from 0.1 to 1.0. Fig. 6 show that with the increasing value of ω, more communication and migration time are needed. From the result, we can see that DDMD incurs more communication overhead than DDMC. The reason is that DDMD needs exchanging state information between individual servers, while in DDMC, broadcasting is avoided in that each server only needs to report its state information to the central controller. However, the cost difference is not large, and compared to the efficiency benefit, DDMD is also an attractive alternative, especially for highly distributed settings, where communication cost is high and reliability is a top priority.

5 Conclusion

In this paper, we propose a series of policies to address the response efficiency challenge in the distributed query-intensive data environment. The main goal is to reduce the query response time by dynamically adjusting data distribution and achieve balanced load. To effectively adapt the system changes, two dynamic policies DDMC and DDMD are designed to adapt for the changing query load and obtain load balance through data migration. DDMC uses a central controller to make migration decisions based on the global load information of the system, while in DDMD, individual server makes decisions based on their own viewpoints of the system without centralized control. Our experiments show that DDMC and DDMD exhibit similar performance in term of average query response time, and as expected, DDMD involves higher communication overhead than DDMC, but the difference is not significant. The experimental results also show that the proposed policies offer favorable response time with increasing query load system-wide.

References

1. Narendran, B., Rangarajan, S., Yajnik, S.: Data distribution algorithms for load balanced fault-tolerant Webaccess. In: Proceedings of The Sixteenth Symposium on Reliable Distributed Systems, pp. 97–106 (1997)
2. Savio, S.: Approximate Algorithms for Document Placement in Distributed Web Servers. IEEE Transactions on Software Engineering, 100–106 (2004)
3. Yokota, H., Kanemasa, Y., Miyazaki, J.: Fat-Btree: An Update-Conscious Parallel Directory Structure. In: International Conference on Data Engineering (ICDE), pp. 448–457 (1999)
4. Lee, M.-L., Kitsuregawa, M., Ooi, B.-C., Tan, K.-L., Mondal, A.: Towards Self-Tuning Data Placement in Parallel Database Systems. In: International Conference on Management of Data (SIGMOD), pp. 225–236 (2000)
5. Feelifl, H., Kitsuregawa, M., Ooi, B.-C.: A fast convergence technique for online heat-balancing of btree indexed database over shared-nothing parallel systems. In: Ibrahim, M., Küng, J., Revell, N. (eds.) DEXA 2000. LNCS, vol. 1873, pp. 846–858. Springer, Heidelberg (2000)

6. Watanabe, A., Yokota, H.: Adaptive Lapped Declustering: A Highly Available Data-Placement Method Balancing Access Load and Space Utilization. In: International Conference on Data Engineering (ICDE), pp. 828–839 (2005)
7. Feldmann, M., Rissen, J.P.: GSM Network Systems and Overall System Integration. Electrical Communication (1993)
8. Jain, R.: The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley Interscience, Hoboken (1991)
9. Wang, T., Yang, B., Gao, J., Yang, D.: Effective data distribution and reallocation strategies for fast query response in distributed query-intensive data environments. In: Zhang, Y., Yu, G., Bertino, E., Xu, G. (eds.) APWeb 2008. LNCS, vol. 4976, pp. 548–559. Springer, Heidelberg (2008)