



ELSEVIER

Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

Data & Knowledge Engineering 51 (2004) 295–323

DATA &
KNOWLEDGE
ENGINEERING

www.elsevier.com/locate/datak

On demand synchronization and load distribution for database grid-based Web applications

Wen-Syan Li ^{*,1}, Kemal Altintas, Murat Kantarcioğlu

NEC Laboratories America, Inc., C&C Research Laboratories, 10080 North Wolfe Road, Suite SW3-350, Cupertino, CA 95014, USA

Received 18 September 2003; received in revised form 9 January 2004; accepted 12 May 2004

Available online 15 June 2004

Abstract

With the availability of content delivery networks (CDN), many database-driven Web applications rely on data centers that host applications and database contents for better performance and higher reliability. However, it raises additional issues associated with database/data center synchronization, query/transaction routing, load balancing, and application result correctness/precision. In this paper, we investigate the issues in the context of data center synchronization for such load and precision critical Web applications in a distributed data center infrastructure. We develop a scalable scheme for adaptive synchronization of data centers to maintain the load and application precision requirements. A prototype has been built for the evaluation of the proposed scheme. The experimental results show the effectiveness of the proposed scheme in maintaining both application result precision and load distribution; adapting to traffic patterns and system capacity limits.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Data center; Database replication; Database grid; DB Cache; Application distribution network

* Corresponding author.

E-mail addresses: wen@sv.nec-labs.com (W.-S. Li), kemal@sv.nec-labs.com (K. Altintas), kanmurat@sv.nec-labs.com (M. Kantarcioğlu).

¹ This work was performed when the author was with NEC Laboratories America, Inc. He is currently affiliated with IBM Almaden Research Center and can be reached at wen@almaden.ibm.com.

1. Introduction

Applying acceleration solutions for Web applications and content distribution has received a lot of attention in the Web and database communities. At the same time, many database and application server vendors are beginning to integrate Web acceleration through data caching in their software. Examples include Oracle 9i [13] which features a suite of application server, Web server, and data cache for deployment at data centers for accelerating the delivery of dynamic content. On the other hand, Akamai [6] extended the concept of content distribution network (CDN) to application distribution network (ADN) by caching and executing stand alone and self-contained J2EE-based applications (e.g. EJB) at edge servers.

Wide-area database replication technologies and availability of data centers allow database copies to be distributed across the network. The goal of this approach is to offset the high cost of replica synchronization by moving data closer to the users (similar to caching in which data is moved closer to the users reducing network latency). Such data center architectures make it more flexible to deploy a “distributed” Web site. However, the system architecture of data center-hosted Web applications has many drawbacks that prevent it from being considered for many e-commerce applications, in which content freshness, request response time, application result precision, and transaction throughput need to be maintained at a very high level and sometimes they need to be guaranteed to be within an acceptable threshold specified as a part of QoS agreement.

In this paper, we investigate the issues of data center synchronization and query routing for load and precision critical Web applications. The complexity arises when all of these load and precision sensitive requirements need to be met in all data centers as well as estimating application errors in response to database content changes. We develop an adaptive data center synchronization technique to effectively coordinate data center synchronization and request routing for load balancing. We have extensively tested our technique. The experiments show the effectiveness and adaptiveness of our solution in maintaining both application result precisions and response time for transactions as a part of QoS specification. We also conducted evaluation on the scalability of our algorithm on an application distribution network (ADN) with 15 data centers and the experimental results show that our algorithm is able to schedule just in time synchronizations and dynamic load distribution even when the load reaches close to 90% of overall system capacity.

Compared with most existing work that focuses on either minimizing response time or achieving QoS (i.e. Quality of Service) for response time, this paper focuses on achieving the following goals at the same time:

- QoS for response time through load distribution; and
- QoS for application result error threshold through on demand synchronization.

The rest of the paper is organized as follows: In Section 2, we describe the current system architecture of data center-hosted applications and point out their drawbacks. In Section 3, we present our solution to these drawbacks on top of existing data center architecture for supporting load and precision sensitive Web applications. In Section 4, we describe how to estimate application errors. In Section 5, we give details of the proposed adaptive data center synchronization scheme. In Section 6, we present the results of experiments and give our analysis. In Section 9, we

summarize related work in the field and compare them with our work. Finally, we give our concluding remarks in Section 10.

2. System architecture of data center-hosted Web applications

Wide-area database replication technology and availability of data centers allows database copies to be distributed across the network. The goal of this approach is to reduce network latency and bandwidth usage by moving content closer to the users which in turn will offset the high cost of replica synchronization. A data center-hosted Web application requires a complete e-commerce Web site suite: Web server (WS), application server (AS), and database management system (DBMS) to be distributed along with the database replicas (i.e. DB Cache). The WS/AS/DB Cache suites are hosted in data centers that are strategically located in key network peering points across the Internet. Updates to the database are still handled using a master/slave database configuration and therefore all updates are handled via the master DBMS at the origin site. The scheme for directing user requests to the closest server is the same as what typical CDNs use to redirect user traffic to appropriate servers.

In order to distinguish between the asymmetric functionality of master and slave DBMSs, we refer to the remote DBMS copies as DB Cache since they are basically read-only copies and cannot be updated directly. DB Cache can be a lightweight DBMS without the transaction management system since user updates are not handled at the remote locations. Note that the DB Cache may cache only a subset of the tables in the master database. The list of tables that are mirrored and the frequency with which they are synchronized with the master database are specified by the DBAs (i.e. database administrators). This information, Mirrored Table Specification, is used by DB Cache Synchronizer and Query Router.

The Query Router directs queries from the application server as follows: (1) Forward all transaction queries and non-transaction queries that require access to the tables that are not mirrored to the Master Database. The results of queries and transactions are received by the Query Router and then forwarded to the application server; and (2) Forward all other read-only queries to the DB Cache. The results of queries and transactions are received by the Query Router and then returned to the application server.

Although the described system architecture for data center-hosted Web applications has many advantages, it has the following drawbacks:

- Since the staleness of DB Cache at data centers is not monitored and synchronization is based on a fixed schedule, precision of the Web applications running at the data center is uncertain. This drawback limits the data center architecture to be used for many precision critical or content sensitive applications, such as applications in the financial markets and decision making systems.
- Since all transactions are executed at the Master Database as well as many queries forwarded from the data centers, the Master Database may not ensure the response time of transactions.
- Web applications demanding higher precision require frequent data center synchronization while some applications do not. Since Web applications may be deployed at data centers on

demand, the pre-defined data center synchronization is not practical and not suitable for response time and precision sensitive applications, such as on line decision making systems.

- The current query routing scheme is inefficient because the resource is not consolidated and fully utilized especially since the traffic pattern at each data center can vary greatly. For example, one data center may be overloaded while the other data center is under-utilized due to dramatically different geographical locations.

3. Adaptive data center synchronization

To address the drawbacks of the existing data center architecture and to support response time and precision sensitive applications, we propose an architecture that aims at building an efficient solution for synchronizing load and precision critical Web applications on top of an existing commercially available data center-hosted Web application system architecture.

In supporting content sensitive applications, each application is expected to specify the error bound threshold within which application errors must be maintained. The errors in application results occur due to the staleness or inaccuracy of data input to the applications. For Web applications hosted at the data center, the errors occur since the state of DB Cache content does not reflect recent changes that have occurred at the master database.

Two possible ways to support content sensitive applications in the distributed data center environment are: First, all queries and transactions are answered and executed at the master database. However, this approach would suffer from poor response time due to network latency and high processing latency incurred due to the heavy load concentrated at the master database. One advantage is that the errors of all applications can be kept within the specified thresholds. Secondly, all transactions are executed at the master database whereas all queries are answered at the remote data centers. To ensure that the errors of all applications are kept within the specified thresholds, all database changes need to be propagated to the data centers immediately. However, this approach would suffer from heavy overhead due to database synchronization across the network.

3.1. Proposed system architecture to support load and precision sensitive applications

We propose a new approach of adaptive data center synchronization specifically for load and precision critical applications in this paper. The architecture (shown in Fig. 1) contains the following new components and files:

- A new component, *Adaptive Load and Precision Critical Synchronizer (ALPCS)*, is deployed to replace the previous DB Cache Synchronizer. The ALPCS is responsible for coordination of data center synchronization and query redirection (i.e. query routing between data centers).
- A new component, *Application Precision Estimator*, is deployed at each data center. The Application Precision Estimator estimates application errors due to data center content staleness based on database log history. Based on the estimated errors and the Application Precision Requirement Specification, the Application Precision Estimator maintains the Query Routing

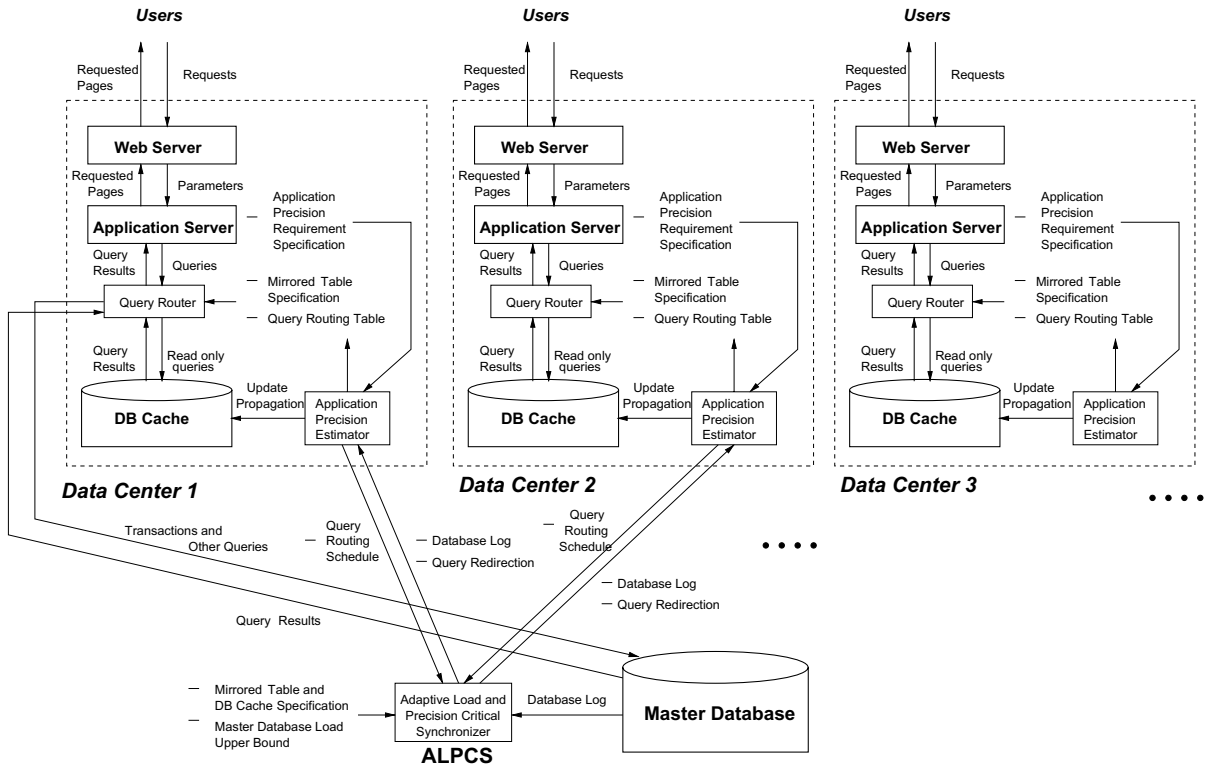


Fig. 1. Architecture supporting load and precision critical Web applications.

Table that is used by the Query Router.

- A new operational file, *Application Precision Requirement Specification*, is needed to represent the threshold of application errors (i.e. upper bound) that must be maintained for all precision sensitive applications hosted at the data centers. The Application Precision Requirement Specification is specified as percentage in terms of the number of tuples in the result set.
- The *Query Routing Table* is extended to track the list of query types and applications that cannot be answered at the data center within their application precision requirement and where to route the queries in case of redirection.

The ALPCS does *not* perform database synchronization across the network directly. Instead, it ships the database update log to the Application Precision Estimator. After receiving the database log, the Application Precision Estimator performs database update propagation to synchronize the DB Cache and the master database. At the same time, the database log is also used by the Application Precision Estimator to estimate application errors and construct the Query Routing Schedule. Examples of the Query Routing Schedule are shown in Fig. 2. The Query Routing Schedule is the schedule when a given application’s requirement is no longer satisfiable at the data center. Note that a snapshot of the Query Routing Schedule is the Query Routing Table.

The Query Routing Schedule of each data center is sent to the ALPCS at the master database. Based on the query routing schedules of all data centers and the load limits on the master

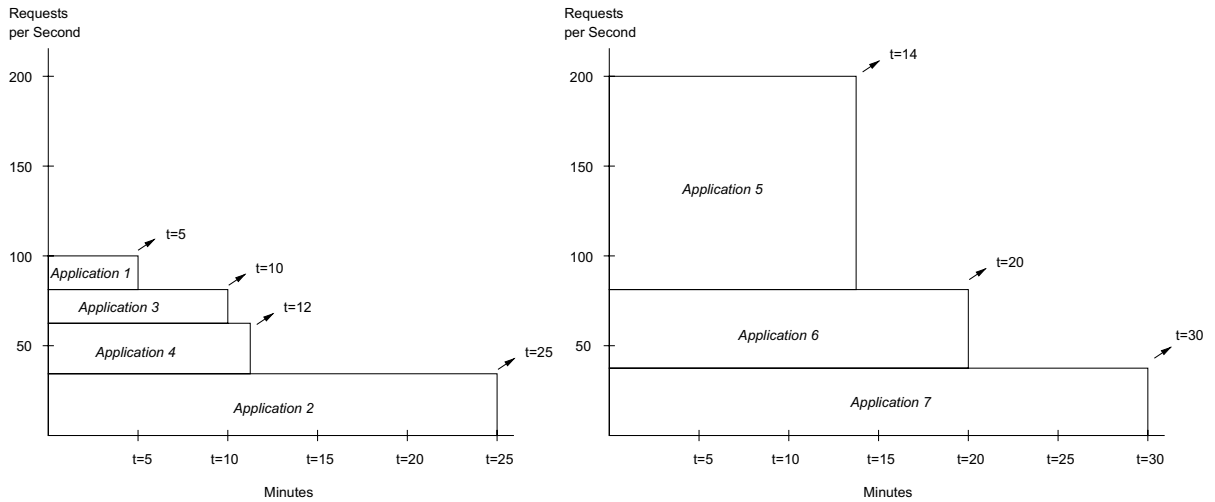


Fig. 2. Query Routing Schedules at Data Center 1 and Data Center 2.

database and data centers (to ensure the response time of transactions executed at the master database), the ALPCS schedules data center synchronization and query redirection from one data center to another if necessary.

Sending the Query Routing Schedule from each data center to the ALPCS is overhead of the proposed scheme. As shown later in Fig. 2, the Query Routing Schedule at a data center can be represented as an array of (*application name, time*) to indicate the time to start forwarding specific application requests to the master database. The overhead is small. Note that sending database update log from the master database to data centers is essential to maintain consistency at the data centers; thus, it should not be considered as overhead.

The details of application precision estimation based on database update signature is described next. The detailed algorithm for the Adaptive Load and Precision Critical Synchronizer will be described in Section 5.

3.2. Query routing schedule

We use few examples to illustrate the concept of query routing schedule that is used as the foundation for the proposed adaptive data center synchronization technique. Assume that there are seven precision critical applications, *Applications 1–4* and *Applications 5–7*, running at *Data Center 1* and *Data Center 2* respectively.

Let there be two relations at the Master Database, namely, R_1 and R_2 . The rates of inserted and deleted tuples for each table at the master database is 0.1% and 0.2% per minute for R_1 and R_2 with respect to the total number of tuples in the two tables respectively. The query arrival rates at *Data Center 1* and the *Data Center 2* are 100 and 200 queries per second, respectively. All transactional operations requested to the data centers are forwarded to the master database. To ensure a fast response time for all transaction operations at the master database, the overall load at the master database (including both queries and transactions) is limited to at most 200 requests

Table 1
Examples of Query Routing Schedule at the *Data Center 1*

Application name	Error threshold (%)	Query percentage (%)	Error estimation function; error	Required refresh frequency (min)
Application 1	0.25	20	$0.5 \times \Delta R_1$	5
Application 2	0.50	30	$0.1 \times \Delta R_2$	25
Application 3	0.75	20	$0.75 \times \Delta R_1$	10
Application 4	1.20	30	$0.2 \times \Delta R_1 + 0.4 \times \Delta R_2$	12

Table 2
Examples of Query Routing Schedules at the *Data Center 2*

Application name	Error threshold (%)	Query percentage (%)	Error estimation function; error	Required refresh frequency (min)
Application 5	0.5	60	$0.35 \times \Delta R_1$	14
Application 6	0.2	20	$0.2 \times \Delta R_1 + 0.15 \times \Delta R_2$	20
Application 7	0.75	20	$0.3 \times \Delta R_2$	30

per second. The functions for estimating errors for each application are given in Tables 1 and 2, where ΔR_1 and ΔR_2 is the percentage of content change in the relation R_1 and R_2 respectively in terms of tuples. Note that the application errors vary with respect to the database content updates.

For a given application error threshold, a given error estimation function (based on DB Cache staleness), and the master database content change rate, we can calculate the required refresh cycle for each application. For example, the error threshold for *Application 1* is 0.25% and the application error increases 0.05% per minute as the content in the R_1 changes. Note that 0.05% is a coefficient impacted by the query statement syntax and value distribution in the tables and table update log. The larger the value is, the more sensitive the applications are to the database content changes. Detailed discussion will be given in Section 4.

In order to keep this application running at *Data Center 1*, the *Data Center 1* must be synchronized with Master Database every 5 min. After we calculate all required refresh frequencies, we can construct a Query Routing Schedule (used by the ALPCS) and a Query Routing Table (used by the Query Router). The Query Routing Schedules for *Data Center 1* and *Data Center 2* are shown in Fig. 2. As we can see, after a data center is synchronized, its utilization rate gradually decreases as time elapses. The utilization rate, however, becomes 100% immediately after the data center is synchronized.

The Query Routing Schedules are sent to the ALPCS at the Master Database periodically. The ALPCS combines the Query Routing Schedules from all associated data centers to construct a Master Database Load Schedule that shows the load at the Master Database in the near future. The Master Database Load Schedule based on the Query Routing Schedules in Fig. 2 is shown in Fig. 3.

Note that there is a load limit at the Master Database as shown in Fig. 3. Based on the Master Database Load Schedule, the ALPCS knows that the load at the Master Database will exceed the limit at the point when $t = 14$ min. Since transactions must be executed at the Master Database,

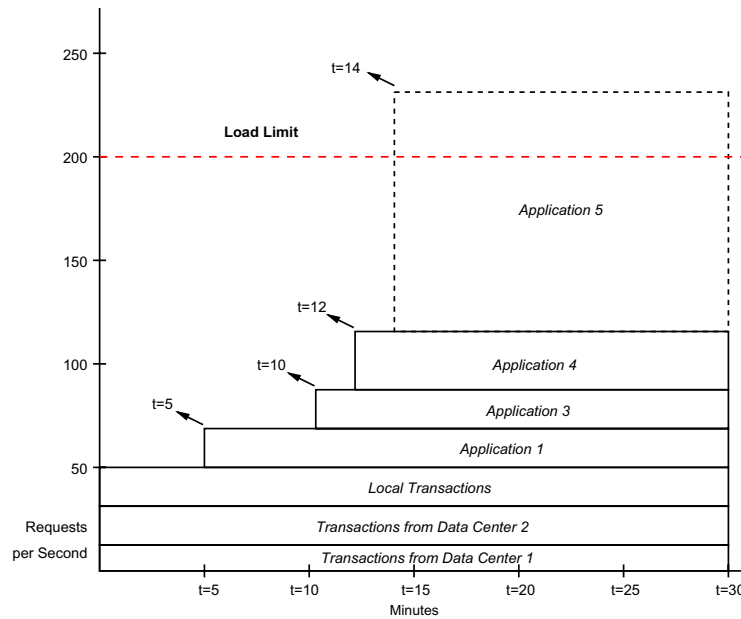


Fig. 3. Load Schedule and limit at the *Master Database*.

one or more data centers must be scheduled to be synchronized so that some queries that might be forwarded from the data centers to the Master Database can now be handled at the data centers. However, the scheduling of data center synchronization is not straightforward. One reason is that such scheduling problem is an NP-hard problem (see Section 9). The complexity arises since the Master Database Load Schedule is impacted by many parameters, including

- the database content change rates at the master database;
- the number of transactions forwarded to the master database from the data centers;
- the number of transactions for the master database itself;
- the query arrival rates and query distribution at each data center;
- the time since last data center synchronization; and
- application error thresholds.

Sometimes there are multiple choices for scheduling. For example, in Fig. 3, we can synchronize either *Data Center 1* or *Data Center 2*. When one of the above parameters change, a new load schedule and a data center synchronization schedule need to be re-calculated. In the example, if we decide to select *Data Center 1* to synchronize and estimate that the synchronization requires 2 min to complete, the synchronization needs to be started at $T = 12$ so that the load forwarded from *Data Center 1* can be removed before $T = 14$. The load schedule at the master database can be re-calculated as shown in Fig. 4. The load labeled with Sync indicates the overhead of synchronization.

As shown in Fig. 4, even we complete the synchronization of *Data Center 1* before $T = 14$, the master database will be overloaded at $T = 25$ based on the new load schedule. Again, we can

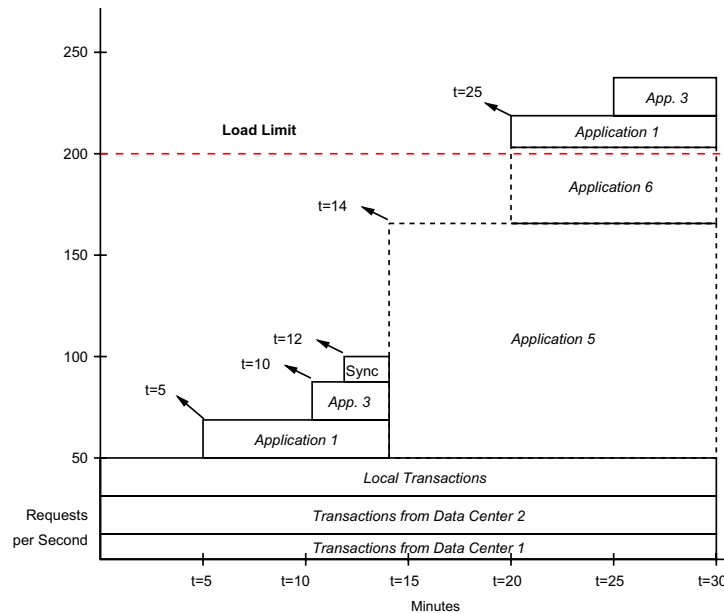


Fig. 4. Load Schedule at the Master Database after synchronization of Data Center 1 scheduled at $t = 12$.

synchronize either Data Center 1 or Data Center 2. If we select Data Center 1 to synchronize, the synchronization needs to started at $T = 22$. If we select Data Center 2 to synchronize and it requires 8 min to complete, the synchronization of Data Center 2 needs to be started at $T = 17$. However, if the synchronization of Data Center 2 requires 12 min, the synchronization of Data Center 2 needs to be started at $T = 13$. As a result, it will overlap with a pre-scheduled synchronization of Data Center 1. To resolve this problem, we will start the synchronization of Data Center 1 earlier at $T = 11$ to avoid concurrent synchronization tasks.

4. Estimating precision of application results based on statistics

In order to correctly schedule our updates, we need to predict the difference between the actual query result (Q_a) (i.e. the result of the query that is run on the master) and the current query result (Q_c) that is evaluated using the tables stored on the data center. In order to achieve these goals, we investigate various approaches and evaluate them experimentally. We describe two major approaches.

4.1. Approaches based on histograms

Histograms are frequently used for estimation of query result set size as surveyed in [18]. We have investigated the possibility of adapting them to estimate application errors caused by stale database tables. Consider the following scenario: assume that we have two relations $R_1(a, b)$ and

$R_2(a, c)$. Let us further assume that $R_1 = \{(a_0, b_0), (a_1, b_1), (a_1, b_2), \dots, (a_1, b_n)\}$ and $R_2 = \{(a_1, c_0), (a_0, c_1), (a_0, c_2), \dots, (a_0, c_n)\}$. Also assume that each table has only one histogram available on the attribute a . Based on user demands, we need to use the following distance function as an error metric based on symmetric set difference.

$$d(Q_a, Q_c) = \frac{|Q_a - Q_c| + |Q_c - Q_a|}{|Q_a \cup Q_c|}$$

where Q_a and Q_c are two query result sets. The tuples are considered equal if every attribute of the tuple is equal.

In such an environment, R_1 and R_2 are mirrored at a data center. It is clear that if we calculate $R_1 \bowtie R_2$, the result will have $2n$ tuples. Now assume that master database has a modification query on R_1 which replaces the tuple (a_0, b_0) with (a_0, b_1) . Although nothing has changed in the histogram tables (i.e. the size of the query) entire query result is different compared to the previous query result (no tuple with b_0 values any more) and at the same time, according to the above distance function the distance between the query results on the master and on the mirror is 100%.

In order to solve this problem, we need to maintain many different histograms which in turn may bring additional overhead. In addition, most of DBMSs do not provide external access to database table histograms, although they are used internally for query optimization.

4.2. Using time-series analysis to predict errors

For the reasons discussed above, we propose using time-series analysis to measure the error, which is a “black box” analysis. In our current implementation, we use the “double exponential smoothing” method although many other time-series prediction methods may be suitable for the task as well.

Given time series y_i , double exponential smoothing method uses two equations to predict the future y_i values.

$$S_t = \alpha y_t + (1 - \alpha)(S_{t-1} + b_{t-1}) \quad (1)$$

$$b_t = \beta(S_t - S_{t-1}) + (1 - \beta)b_{t-1} \quad (2)$$

where y_t is the last value observed, α and β are coefficients that $0 < \alpha, \beta < 1$. S_1 is usually set as y_1 and b_1 can be initialized as $y_2 - y_1$.

In order to use the “double exponential smoothing”, we will create time-series data that corresponds to an average error per unit time as follows: when all the tables are synchronized at a data center at time t_2 , the data center will predict the current average error as $e_i = d(Q_{t_i}, Q_{t_{i-1}})/(t_i - t_{i-1})$. It is easy to replace the above y_i with e_i . Given the tolerated error bound err , we can estimate when some of the tables at the data center need to be synchronized as $\frac{err}{S_t + b_t}$ where t_i is the last synchronization time.

In order to safeguard against extreme predictions, we make sure that the data center is updated at least every t_{max} time after its last update in case that the master database experiences an unexpectedly huge load.

We have implemented the above time-series analysis to estimate application errors and incorporated it into the data center architecture. The procedure is as follows:

1. Receive the database log from the *Master Database*.
2. Calculate the interval between current time and the previous synchronization time, t , as T .
3. Incorporate the database log to the *DB Cache* incrementally to perform sampling for time-series analysis. For example, if we plan to take 10 samples in the interval T , the first synchronization processes database log with a time stamp between t and $t + T/10$ in the order of log time stamps. And, the second synchronization processes database log with a time stamp between $t + T/10$ and $t + 2 \times T/10$.
4. Calculate the difference between query results using the stale *DB Cache* content and partially synchronized *DB Cache* content. The difference is the application error. And, after all data-based update log is incorporated into the *DB Cache*, we have constructed a map (based on history) between application errors and changes in relations.
5. Such a map will be used to estimate application errors until the next synchronization.

4.3. Experimental analysis

To evaluate the correctness of the proposed time-series analysis approach, we have conducted two experiments for evaluation. We created two relations $R_1(a, b)$, $R_2(a, c)$ where a, b, c are integers with range (0, 1000). Each relation has 10,000 tuples initially. Our goal is to predict the update times for the data center such that the symmetric difference $\left(\frac{|Q_a - Q_c| + |Q_c - Q_a|}{|Q_a \cup Q_c|}\right)$ is below some certain error threshold.

We ran experiments to see the error rate of different query types in similar update setting. We ran five queries where

$Q_1 = \text{select } * \text{ from } R_1 \text{ where } R_1 \cdot b < 200$

$Q_2 = \text{select } * \text{ from } R_1 \text{ where } R_1 \cdot b < 300$

$Q_3 = \text{select } * \text{ from } R_1 \text{ where } R_1 \cdot b < 400$

$Q_4 = \text{select } * \text{ from } R_1, R_2 \text{ where } R_1 \cdot a = R_2 \cdot a \text{ and } R_1 \cdot a \leq 100 \text{ and } R_1 \cdot a \geq 20$

$Q_5 = \text{select } * \text{ from } R_1, R_2 \text{ where } R_1 \cdot a = R_2 \cdot a \text{ and } R_1 \cdot a \leq 250 \text{ and } R_1 \cdot a \geq 200$

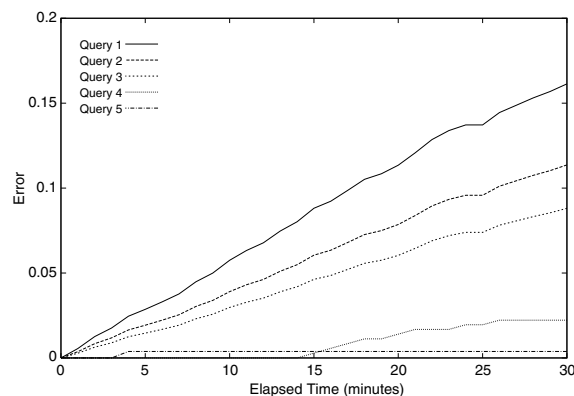


Fig. 5. Error vs time graph for query 1 to query 5.

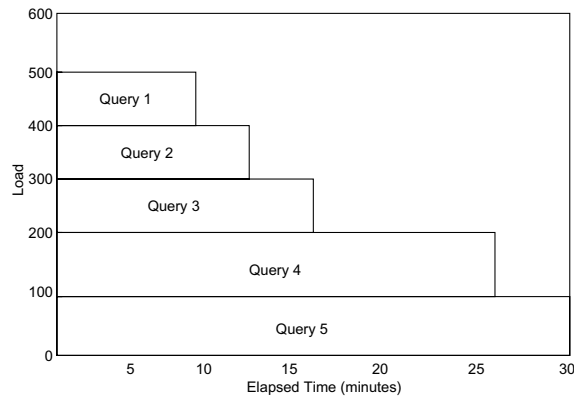


Fig. 6. Corresponding Query Routing Schedule for Fig. 5.

The application (i.e. queries) error rates with respect to database table changes are given in Fig. 5. With the error bounds for all five queries set as 0.05, we can estimate when the application errors will be out of bounds and the applications need to be forwarded to the master database. The query routing schedule for the data center can be derived as shown in Fig. 6.

5. Adaptive synchronization of data centers and load distribution

In Section 3, we presented the system architecture for data center-hosted applications with the deployment of the proposed adaptive data center synchronization to ensure response time and application precision guarantees. In this section, we describe the adaptive data center synchronization and load distribution techniques in more detail.

5.1. Load distribution guideline

We set the guideline for load distribution and load limit of the master database and the data centers as follows:

- Transaction requests receive the highest priority to be executed. Thus, all transactions must be forwarded to the master database and the load limit on the master database must be maintained.
- Read requests that are answered within their specified error bounds receive the second highest priority.
- Read requests that are answered within their desired response time receive the third highest priority.

Based on these guideline, the data centers will forward all requests that cannot be answered locally because error bounds or load limits are reached at the master database. However, the master database accepts transactions first. If the master has sufficient capacity, it will accept

requests forwarded from the data centers for which error bound is maintained. Then, if the master database still has sufficient capacity, it will accept requests forwarded from the data centers solely because load limits are reached at the data centers. However, if the master database's load limit is reached, it has a choice to redirect some read-only requests forwarded from one data center to another data center which has the tables to answer the requests and is under-utilized.

5.2. Data center synchronization scheduling

One of the main functions of the data centers is to serve as many requests as possible so that the load at the master database is kept below the limit in order to process transactions in fast response time. However, the load of the master database will continue to increase as time goes by since more and more applications will no longer be able to execute at the data centers once their error bounds are reached. The master database must synchronize data centers just in time so that some requests forwarded from data centers can be removed from the master database.

The algorithm for adaptive data center synchronization (ALPCS) is as follows:

1. After synchronizing a data center, the data center constructs a new query routing schedule and sends it to the ALPCS at the master database.
2. Based on the query routing schedules from all the data centers, the ALPCS constructs the load schedule for the master database.
3. ALPCS looks into the future in the load schedule of the master database and finds the first point that the master database will overflow. ALPCS does take into account the following factors:
 - synchronization cost (i.e. overhead added to the load at the data center and the master database during the synchronization); and
 - time required to complete synchronization which is impacted by the interval between two synchronization, database update rates, and synchronization speed.
4. By looking up the master database load schedule, ALPCS selects and schedules one data center for synchronization which will offset the largest amount of load from the master database just before it overflows.
5. Reconstruct the master database load schedule based on the reduced loads as a result of the previous synchronization.
6. Look up the new master database load schedule, select and schedule a data center for synchronization which will offset the largest amount of load from the master database. If the newly scheduled database synchronization overlaps with existing scheduled synchronization task, move the existing scheduled synchronization task forward. If the rescheduling of the existing scheduled synchronization task overlaps with another existing scheduled synchronization task, repeat the reschedule forward recursively until there is no overlap.
7. If the overlapped synchronization tasks cannot be rescheduled (i.e. the master database will be overflowed), locate under-utilized data centers to redirect the requests. Note that the selected under-utilized data centers must have all the tables to run the forwarded requests/applications.
8. If under-utilized data centers cannot be found, reject some forwarded requests from data centers based on the above load distribution guideline.

Because there may be many foreign key constraints between database tables, partial synchronization poses additional complication and may cause database inconsistency. In this paper, we assume that all tables in the data center are updated in one synchronization.

In order to calculate the expected life time (the duration where freshness requirements are still satisfied) for each application, we need to maintain a constant memory and each new prediction requires only constant operation. Therefore if there are total A applications running in the system, we only need $O(A)$ time to calculate the expected life times. At the master database, based on the expected life times and the access frequencies of applications, we need to find an appropriate data center to be synchronized. In our algorithm, we are creating a schedule by choosing the first data center which reduces the load most at the master database just before the overloading of the master database. Even a naive implementation will take at most $O(A \cdot D)$ where D is the number of DCs.

6. Experimental evaluations

We have conducted a comprehensive set of experiments to evaluate the proposed system architecture and the *Adaptive Load and Precision Critical Scheduler* (ALPCS) algorithm. In this section, we describe the experimental results. We start with the settings for the experiments.

6.1. Experiment setting

We prototyped a test environment consisting of one master database and three data centers. There are a total of 30 applications in the system that access 20 tables. Among these 30 applications, 15 of them access only one table, 10 of them access 2 tables, and 5 of them access 3 tables. The tables are randomly assigned to each application.

Each data center is installed with 20 applications. The applications running on each data center are randomly assigned. Table update rates are between 0.01% and 0.05% per minute. We assume that the application error function is a coefficient C multiplied by the geometric mean of table change rates for all relations accessed by the application. C is a coefficient between 0.2 and 0.8. The error bound for each application is assigned randomly a value between 0.5% and 1.5%.

6.2. Experiments

In the first experiment, we have imposed a load limit of 500 to the master database (MDB) and the data centers have no imposed load limit. The number of requests for data centers DC1, DC2 and DC3 are 400, 350 and 300 respectively. The results of the experiment are shown in Fig. 7. On the left of the figure, we show the load at the master database. Three dotted lines indicate the load forwarded from each data center and the solid line indicates the total load at the master database including the following:

- the local transactions occur at the master database;
- the transactions forwarded from the data centers;
- the requests forwarded from the data centers due to overflows; and

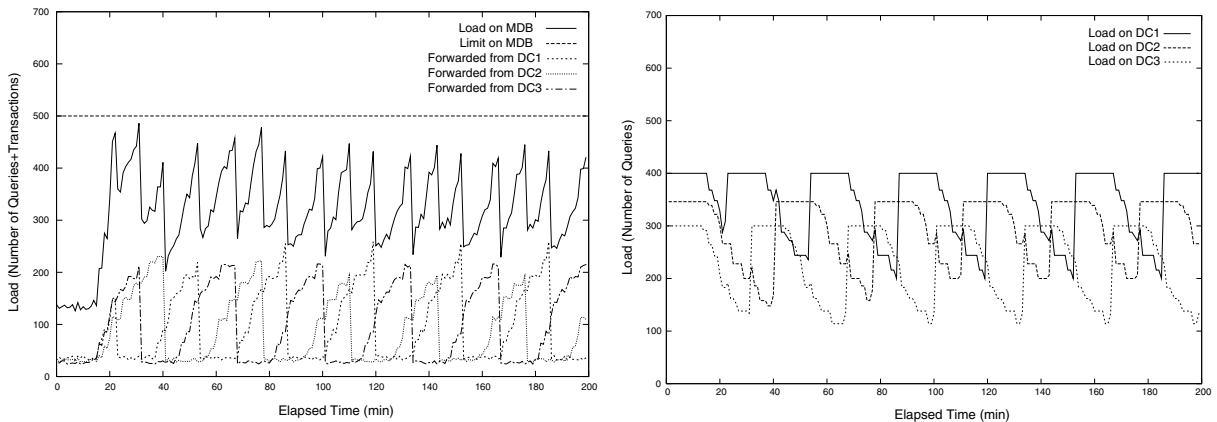


Fig. 7. Load at MDB and data centers (loads at DC1, DC2, DC3: 400, 350, 300, load limit on MDB: 500).

- the requests forwarded from the data centers because applications have reached the specified error bounds.

On the right of the figure, three dotted lines indicate the load at each data center while the solid line indicates the load limit placed on each data center.

Initially, the data centers send their transactions to the master database and for the first few minutes, the master database load remains stable. Then the application error bounds at the data centers start to be exceeded and the queries are forwarded to the master database. Based on the query routing schedules of the data centers, the master database makes predictions about its future load and starts updating the data centers just before it exceeds its load limit. The figures show us that our system can handle the load balancing and the error bound conditions.

In the next experiment, we used the same parameters for our system and in addition we imposed a load limit of 350 for each of the data centers. Thus, whenever the load of a data center exceeds its limit, the additional load is forwarded to the master database. The results of the experiment are shown in Fig. 8. In contrast to the previous experiments, the initial load of the master database now starts from a higher value since each of the data centers is sending some queries due to the load limit in addition to the transactions. The master database again prepares its own synchronization schedule based on the routing schedules of the data centers and updates the data center which will reduce the master database load the most. The figures show us that the system can maintain the load limit and application error bounds both at the master database and at the data centers.

The results of the same experiments after decreasing the data center load limits to 300 are shown in Fig. 9. Now, the master database synchronizes the data centers more frequently and the initial load of the master database is higher due to the additional queries sent from the data centers because of the load limit.

When the load limit of the data centers is decreased to 250, all of the data centers will always be above the load limit. Keeping in mind that some of the capacity is reserved for synchronization cost, the actual limit of the data centers is 230 and 360 queries will be sent to the master database

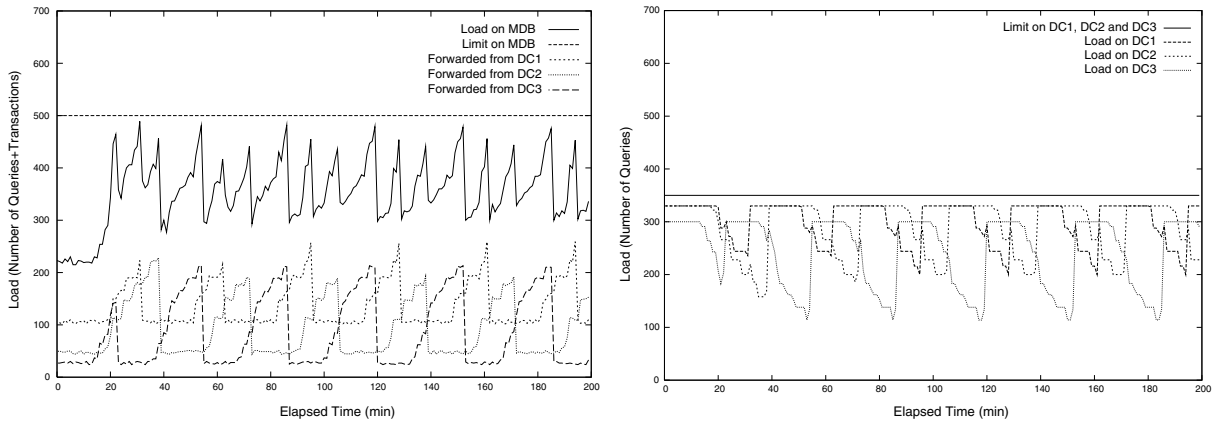


Fig. 8. Load at MDB and data centers (loads at DC1, DC2, DC3: 400, 350, 300; load limit on MDB, DC1, DC2, DC3: 500, 350, 350, 350).

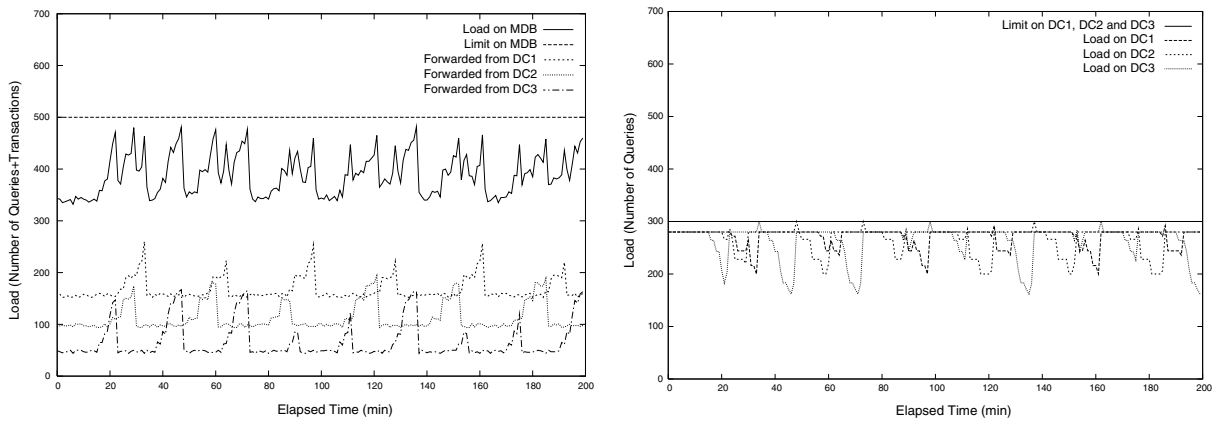


Fig. 9. Load at MDB and data centers (loads at DC1, DC2, DC3: 400, 350, 300; load limit on MDB, DC1, DC2, DC3: 500, 300, 300, 300).

every minute due to the data center load limits. If we consider the master database local transactions in addition to the forwarded transactions, the master database is always at the boundary of its load limit. As it can be seen from Fig. 10, although master database is doing its best to keep the applications and data centers within the bound, the overall system load cannot be handled with the given system capacity. Thus DC1 is always overloaded.

Our algorithm is able to handle cases where the data center load distributions are similar or different provided that the overall load within the system is below the overall system capacity. Fig. 11 shows us a setting where the data center loads and the transaction arrival rates at each data center are the same and the system can easily maintain the error and load bounds. Figs. 12 and 13 show two settings where the data center load distributions are unbalanced. However, our system can still handle these situations.

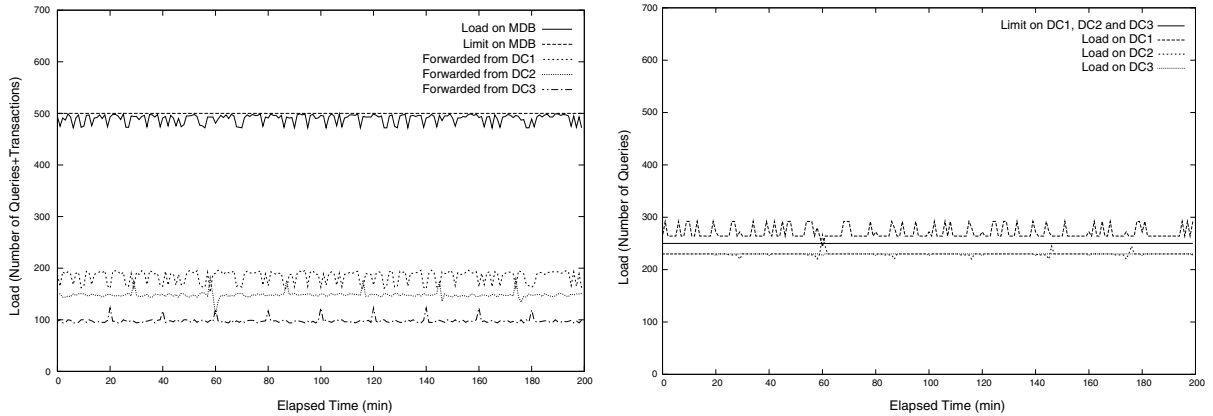


Fig. 10. Load at MDB and data centers (loads at DC1, DC2, DC3: 400, 350, 300; load limit on MDB, DC1, DC2, DC3: 500, 250, 250, 250).

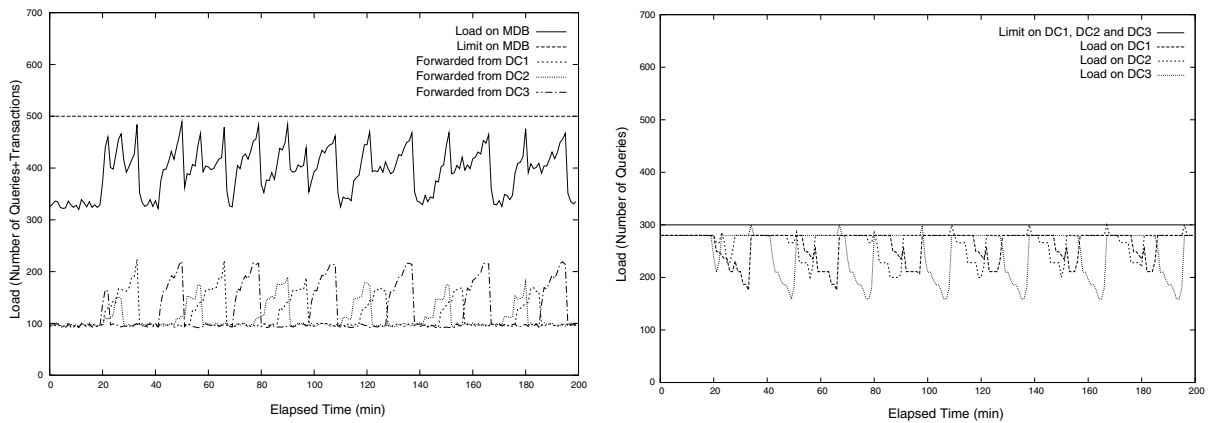


Fig. 11. Load at MDB and data centers (loads at DC1, DC2, DC3: 350, 350, 350; load limit on MDB, DC1, DC2, DC3: 500, 300, 300, 300).

The last experiment we conducted tests the impact of redirecting some requests from one data center to another. In order for one application to be forwarded from one data center to another, the target data center should have all of the tables required by the queries of the application. Also, the data center should be under-utilized, i.e. it should have some permanent resources to run the queries of the application even after the synchronization. Moreover, the application should be able to run at the target data center for some more time before the error bound of the application is exceeded. In our experiment, we used the load values 500, 350, 250 for data centers DC1, DC2 and DC3 respectively. The load limit of the master database is 300 and that of data centers is 350. Note that while data centers DC1 and DC2 are above the limit, DC3 will not be forwarding any queries to the master database due to load limit, so it is under-utilized. The results of the experiment without any application forwarding can be seen in Fig. 14. Although the master database is doing its best to maintain the load limit at DCs, DC1 is always overloaded while DC3

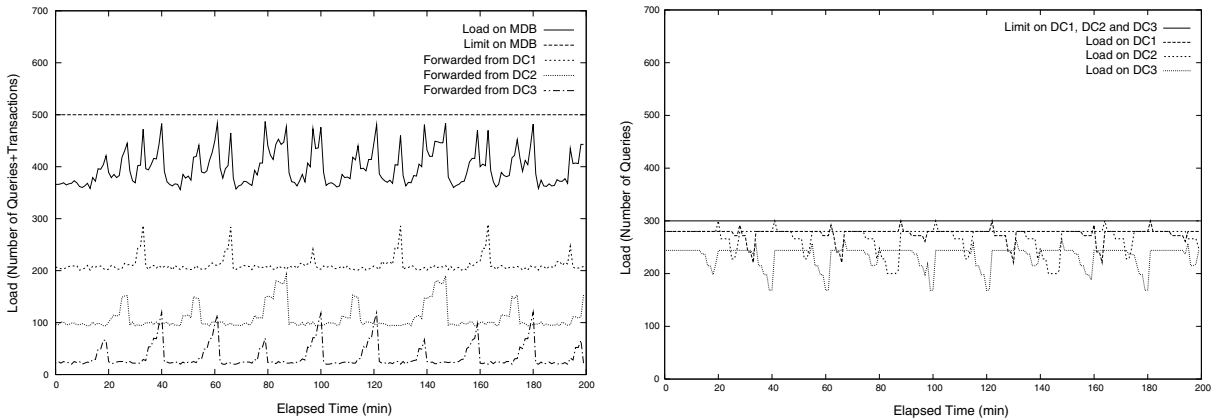


Fig. 12. Load at MDB and data centers (loads at DC1, DC2, DC3: 450, 350, 250; load limit on MDB, DC1, DC2, DC3: 500, 300, 300, 300).

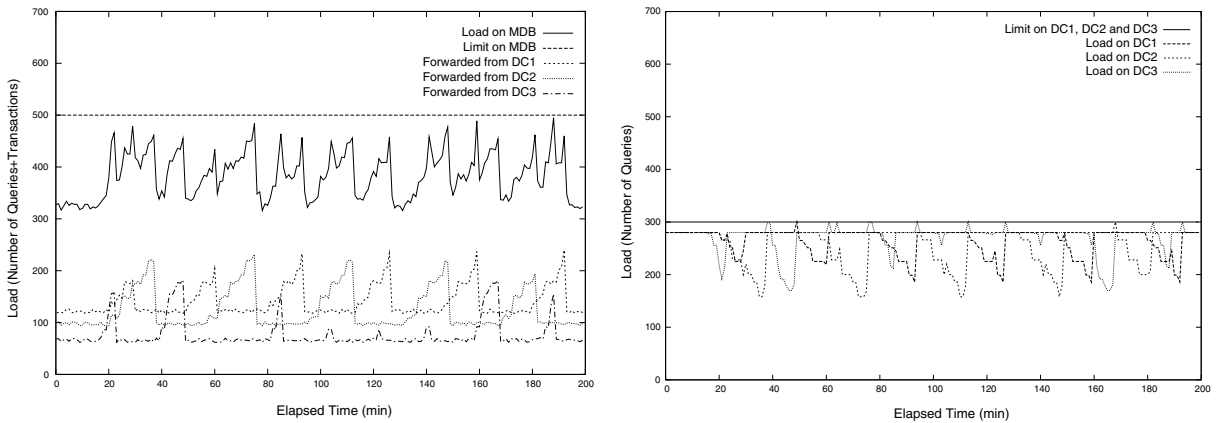


Fig. 13. Load at MDB and data centers (loads at DC1, DC2, DC3: 375, 350, 325; load limit on MDB, DC1, DC2, DC3: 500, 300, 300, 300).

is always under-utilized. The results of the experiment with the same parameters and with application forwarding can be seen in Fig. 15. Here, some of the applications are forwarded from DC1 to DC3, so the available capacity of DC3 due to the under-utilization can be used.

7. Comparisons with other algorithms

The round robin algorithm is one of most frequently deployed scheme for data center synchronization. We observe that the round robin type of data center synchronization schemes are suitable when the variability in the loads, applications capacity, load limits, and other parameters at data centers are low. In particular, the round robin type of data center synchronization schemes do not work when the loads, applications capability, and load limits of data centers are diverge

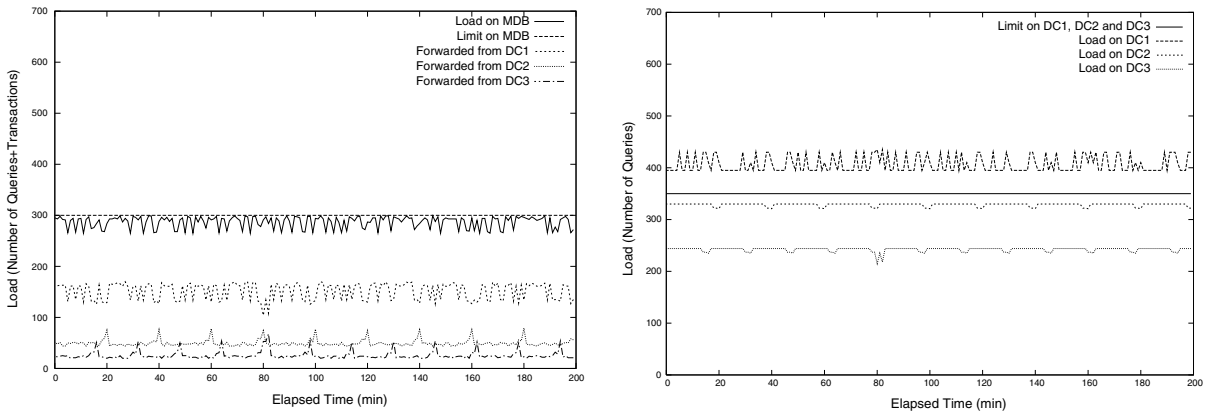


Fig. 14. Load at MDB and data centers (loads at DC1, DC2, DC3: 500, 350, 250; load limit on MDB, DC1, DC2, DC3: 300, 350, 350, 350; with load distribution).

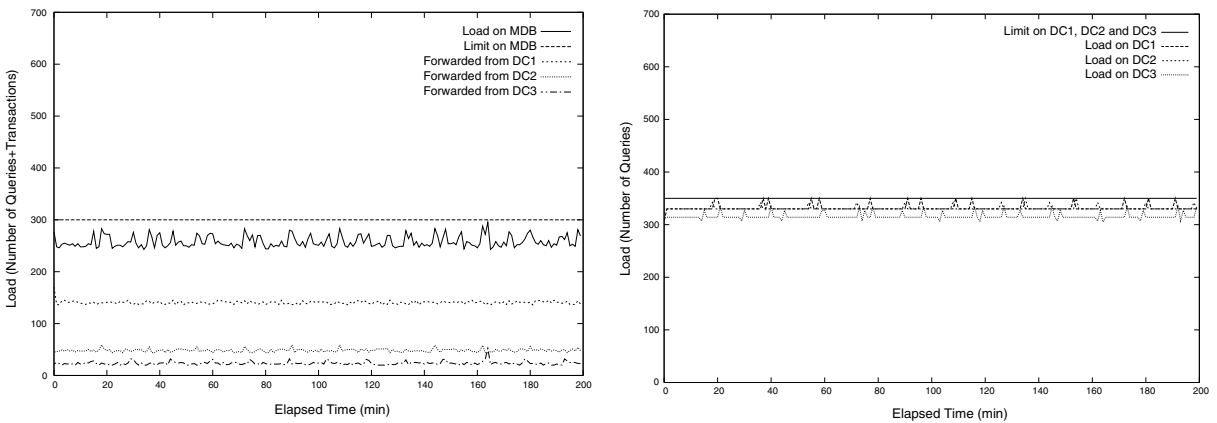


Fig. 15. Load at MDB and data centers (loads at DC1, DC2, DC3: 500, 350, 250; load limit on MDB, DC1, DC2, DC3: 300, 350, 350, 350; with no load distribution).

and change dramatically. For example, the load and deployed applications of one data center is different from that of the other data centers. To verify our observation, we implemented and tested the two variations of round robin algorithms for the comparison purpose:

- the round robin algorithm (RR) in which data center synchronization schedule is fixed and there is load forwarding from the data centers to the master database but there is no load forward from the data centers to other under-utilized data centers; and
- the round robin algorithm with load redirection (RRF) in which data center synchronization schedule is fixed but a load redirection scheme is deployed to distribute load from data centers to the master database and other under-utilized data centers when applications at the data centers reach their error bounds or the load at the master database reaches its limit.

7.1. Experiment setting

As is the case in the previous section, there are 20 tables and 30 applications in the system. Among these 30 applications, 15 of them access only 1 table, 10 of them access 2 tables, and 5 of them access 3 tables. The tables are randomly assigned to each application. We assume that the application error function is a coefficient C multiplied by the geometric mean of table change rates for all relations accessed by the application. C is a coefficient between 0.2 and 0.8.

To verify our observations above, we adjust the table update rates and application error bounds to create unbalanced load in the system. We increase the variance of the application error bounds by assigning randomly a value between 0.5% and 2.5% instead of the previous range of 0.5% and 1.5%. We also increase the table change rates from the range between 0.01% and 0.05% per minute to the range between 0.04% and 0.1% per minute. Note that now the average TTL (i.e. time to live) of applications is shortened and the variance of applications' TTL increase. We sort the 30 applications by TTL and assign 20 applications with the shortest TTL to DC1 and assign the 20 applications with the longest TTL to DC2 and DC3.

To summarize, we have created a very sensitive setting in which the load distribution among the data centers and the master database is dynamic because the table update rate is very fast.

7.2. Experiments

In the first experiment, the load limit of MDB is 500 and that of the data centers is 250. The loads of DC1, DC2 and DC3 are 450, 350, 200 and transactions are 35, 30, 30 respectively. In this setting, the query routing schedules of each data center is quite different so that the required synchronization frequency for each data center is different. We experimented our algorithm against round robin scheduling with the same parameters. Although the load distribution of the data centers are unbalanced, our algorithm can handle the situation by synchronizing the data centers at different frequencies as it can be seen in Fig. 16. However, round robin algorithm has to synchronize the data centers with the same frequency and the same period. Despite the fact that

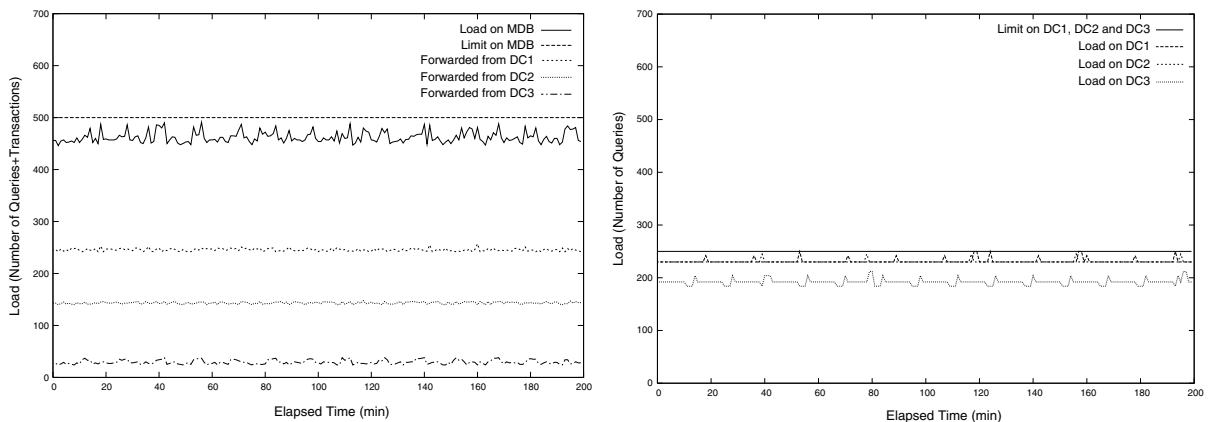


Fig. 16. Load at MDB and data centers with the ALPC algorithm (loads at DC1, DC2, DC3: 450, 350, 200; load limit on MDB, DC1, DC2, DC3: 500, 250, 250, 250).

the total number of synchronizations in both algorithms are the same, RR algorithm cannot handle the situation as it can be seen in Fig. 17. Note that DC1 has the shortest TTL applications and starts sending queries before the other data centers. When DC2 tries to send queries to master database, since it is already occupied by the queries of DC1, queries of DC2 are refused and it overflows.

When we repeat the same experiment with the possibility of load redirection from one data center to the master database as well as to another data center, RRF Scheduling can handle the situation. As it can be seen in Fig. 18, some of the load is transferred from DC1 and DC2 to master database and to DC3 in order to make use of the under-utilized resources in DC3.

In the next experiment, we change the loads on all of the data centers to 350. In this setting, none of the data centers is under-utilized. As the results can be seen in Figs. 20 and 21, the round

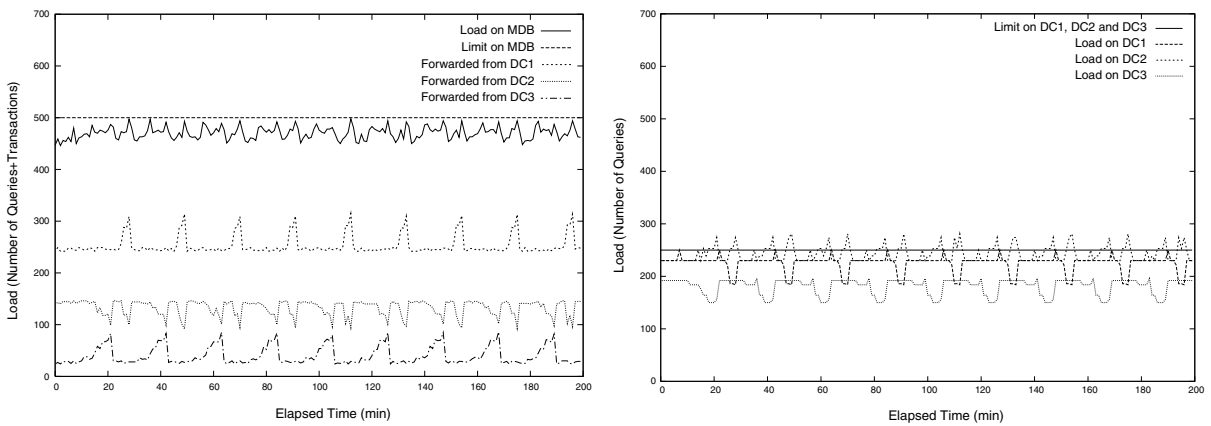


Fig. 17. Load at MDB and data centers with the round robin algorithm (loads at DC1, DC2, DC3: 450, 350, 200; load limit on MDB, DC1, DC2, DC3: 500, 250, 250, 250).

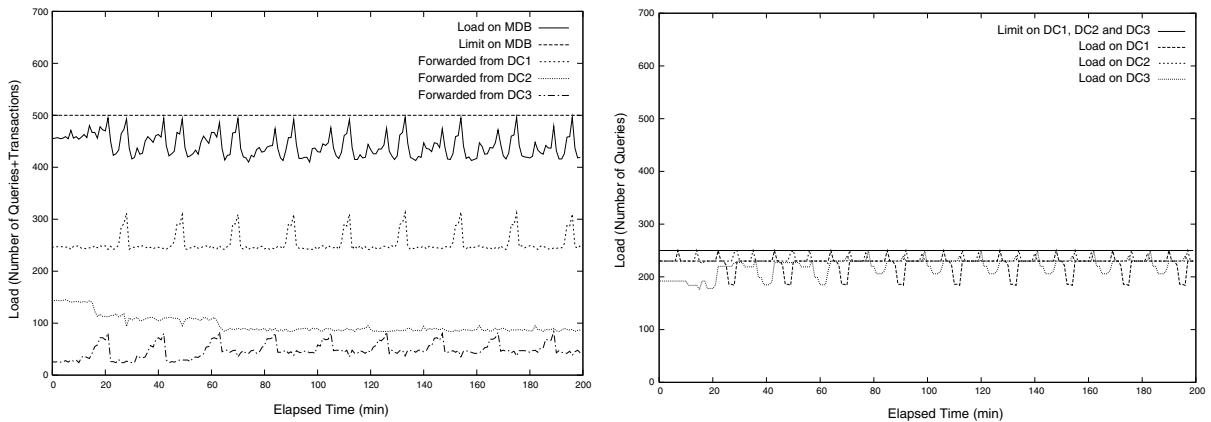


Fig. 18. Load at MDB and data centers with the round robin algorithm with load redirection (loads at DC1, DC2, DC3: 450, 350, 200; load limit on MDB, DC1, DC2, DC3: 500, 250, 250, 250).

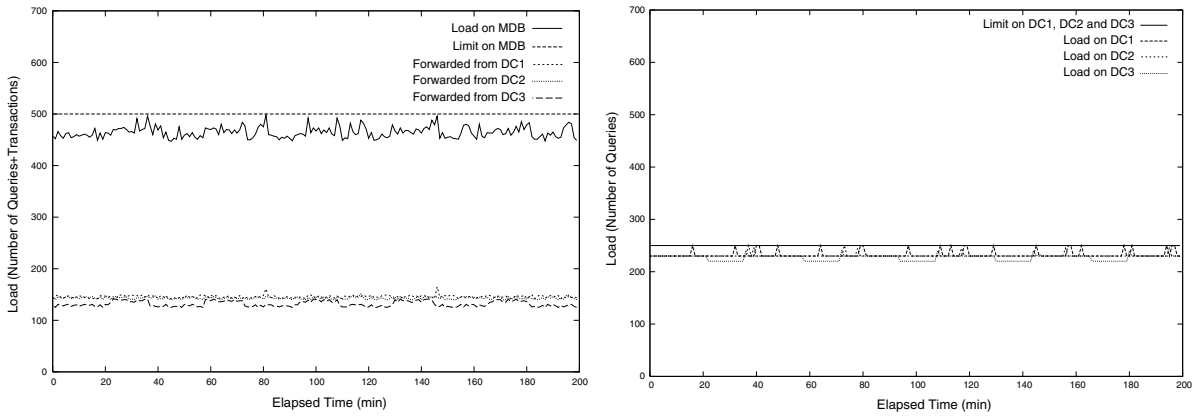


Fig. 19. Load at MDB and data centers with the ALPC algorithm (loads at DC1, DC2, DC3: 350, 350, 350; load limit on MDB, DC1, DC2, DC3: 500, 250, 250, 250).

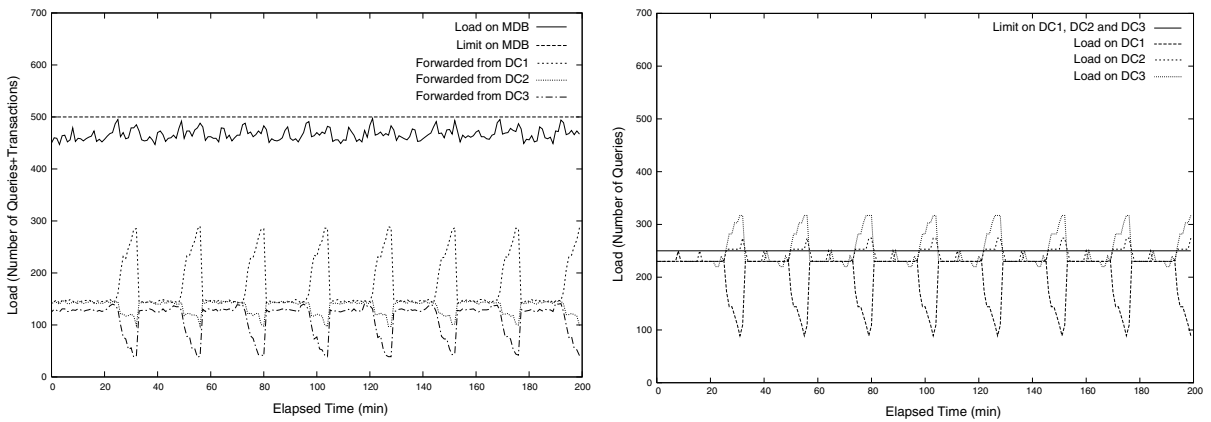


Fig. 20. Load at MDB and data centers with the round robin algorithm (loads at DC1, DC2, DC3: 350, 350, 350; load limit on MDB, DC1, DC2, DC3: 500, 250, 250, 250).

round robin algorithm cannot handle this situation. Although the system is capable of forwarding some of the applications from one data center to the other, since there are not any under-utilized data centers, the system cannot handle the situation without scheduling the synchronizations in a smart way. Our algorithm, on the other hand, can handle the situation by adjusting the frequency of the synchronization for each of the data centers as it can be seen in Fig. 19.

8. Evaluation of scalability

In addition to the experiments in Sections 6 and 7, we also conducted an evaluation of our algorithm on the aspect of scalability in scheduling a large number of data centers; especially when

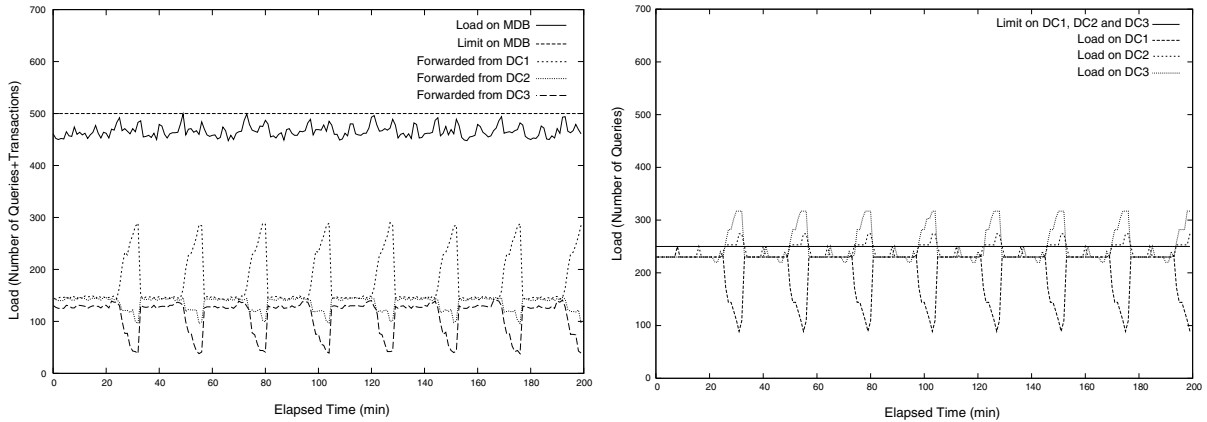


Fig. 21. Load at MDB and data centers with the round robin algorithm with load redirection (loads at DC1, DC2, DC3: 350, 350, 350; load limit on MDB, DC1, DC2, DC3: 500, 250, 250, 250).

the user requests and transactions are close to the overall system capacity. In this section, we describe the experimental results to test on a large scale application delivery network (ADN) of 15 data centers and loads close to 90% of its overall capacity. We start with the settings for the experiments.

8.1. Experiment setting

The overall system has 20 relations with table change rates between 0.01% and 0.05% and 40 applications with error bounds between 0.5% and 1.5%. Fifteen of the applications access a single table, 20 applications access two tables and 5 applications access three tables. The tables are distributed among the applications randomly. Each data center has 20 applications running which are randomly assigned to data centers. We assume that the application error function is a coefficient C multiplied by the geometric mean of table change rates for all relations accessed by the application. C is a coefficient between 0.2 and 0.8.

8.2. Experiments

In this experiment, we have 15 data centers with loads between 200 and 550. The load limit on the master database is 1750 and the load limit on data centers is 350. Note that we have overloaded data centers as well as under-utilized data centers, but the system has enough capability to serve the user requests.

In addition, each data center is forwarding a maximum of 35 transactions per minute. Considering the 40 local transactions of the master database, we can expect a total of around 5900 queries system wide. The theoretical overall system capacity is 7000 queries. Since the master database and data centers will experience additional loads during the synchronizations, each data center and the master database need to reserve a capacity equivalent to 20 queries for synchronizations. The available system capacity is 6680 (i.e. $(7000 - (15+1) \times 20)$). Thus, 4.5% of capacity is reserved for the operational purposes.

The overall load of the data centers is 5400 including requests and transactions, which is 88.3% of the available system capacity. The setting provides a good test case for our algorithm in scheduling and load distribution while maintaining applications within error bounds. The results of the experiment are shown in Fig. 22. On the left of the figure, we show the load at the master

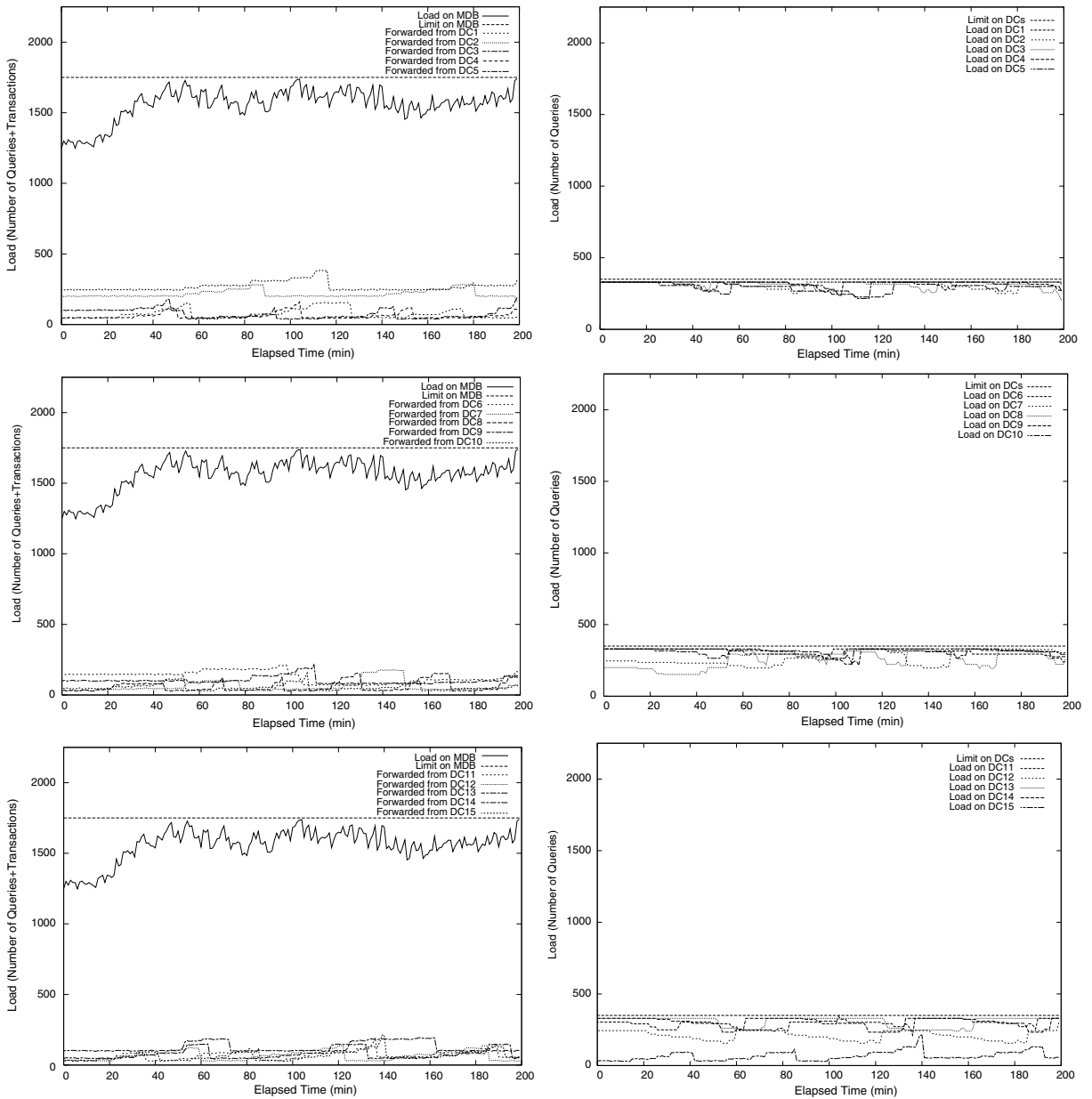


Fig. 22. (left) Load and limit at MDB and loads forwarded from data centers; and (right) loads and limits at data centers.

database and the load forwarded from each data center due to load limits or application error bounds have been reached. On the right of the figure, we show the load at each data center. Since the number of data centers is large, we divided the data centers into three groups in order to have more readable graphs. As it can be seen from the graphs, the master database can handle the situation without overloading any data center while maintaining application precision.

9. Related work

The update scheduling problem in load and precision sensitive systems can be compared to the problem of scheduling tasks on a single machine in order to minimize the weighted completion time under precedence constraints [10], which has been proved to be *NP-hard* for the general case [21].

In [20], the problem of updating Web caches when their back-end databases receive updates is investigated. At the conceptual level, we can view the DCs as Web caches but there is no parallel for the error bounds on queries, since unlike our approach they do not look at levels of freshness. They assume a Boolean measure of freshness i.e. data is either fresh or stale. The authors focus on maintaining the overall quality of answers generated from the cache above an expected threshold value by refreshing the views selectively based on popularity. We assume that the order of updates at the data centers must follow that at the master database, which then prevents us from doing selective updates. Therefore the work done in [20] can be seen as a relaxed version of the work done by us. We intend to relax the update order preservation constraint in our future work so as to increase the speed of synchronization. But doing so would entail additional processing costs both at the Master Database and DB Cache and could affect the overall load experienced by the Master.

Refs. [4,5] look at scheduling of updates in the context of real-time databases, where update operations have to compete with transaction processes that have soft deadlines. We assume that updates seen by the Master Database are a result of successful completion of transactions. We do not look at conflicts arising between transactions due to such updates. Ref. [5] focuses on reducing the cost of refreshing the data without sacrificing data timeliness. We assume the cost of refreshing remote DB Caches as an additional load on the Master database but do not focus explicitly on reducing such costs.

Cho and Garcia-Molina in [12] deal with the issue of when to poll remote sources in order to update local copies and improve database freshness. Their approach is more suited for Web crawlers and systems that access autonomous sources e.g. mediators, whereas we look at tightly knit systems where the source (i.e. Master Database) is fully cooperative and provides information about changes to its underlying data.

Refs. [25,26,30] deal with consistency issues for update propagation in replicated database systems. Even though we model our problem in a replicated database domain, the assumption that all transactions occur only at the Master Database takes care of any issues brought forward in terms of consistency. The data at the remote data centers is used only to answer read-only queries, and all updates at the data centers are done in the order they are seen at the Master database.

Franklin et al. in [14] compared the performance issues of many different cache consistency protocols. In the context of data center-hosted applications, applications that require to maintain the ACID properties are forwarded and executed at the master database. For the applications that do not require strict ACID properties are processed at the data centers for fast response time. Our work actually extends the typical system architecture for data center-hosted Web applications to have a more flexible consistency policy for application results.

Another closely related area to our work is dynamic data replication algorithms. Wolfson et al. in [29] gave a replication algorithm, which establishes a replica of an object at a site if the object is accessed more often at that site than updated at all other sites. Our work can be seen as complementary to this work, in the sense, we are considering how to efficiently update the replicated data objects.

Cached data can effect the query optimization decisions, and efficient query processing may require some data caching. This dependency and efficient integration of query optimization with data caching is given in [15,19]. Again these algorithms do not consider how to update the replicated copies efficiently.

Distribution of replicated data objects are also considered in Web content delivery and peer to peer computing domains. For example, Chen et al. in [11] gave dynamic content distribution system built on top of a peer-to-peer location service. They propose a replica replacement protocols that consider quality of service requirements. In Ref. [16], replication is considered to reduce access time in a peer-to-peer system. Usually similar works in peer-to-peer computing domain considers static data objects. They do not usually consider the effect of running updates and SQL queries on the data objects. Therefore, they do not consider how to propagate the updates efficiently.

Other related works include [1,3], where authors propose a diffusion-based caching protocol that achieves load balancing, [2] which uses meta-information in the cache-hierarchy to improve the hit ratio of the caches, [27] which evaluates the performance of traditional cache hierarchies and provides design principles for scalable cache systems, and [9] which highlights the fact that static client-to-server assignment may not perform well compared to dynamic server assignment or selection. This related work focuses on object level content, such as images, rather than database tables and Web applications.

Caching database tables for improving application response time was addressed in [24]. In their work, they mainly consider simple extensions to existing DB2 to facilitate middle-tier database caching. The work in [7,8] focuses on grouping database tables for mirroring by detecting table dependency, such as foreign key constraints. Their work is complementary to the work described in this paper. On the other hand, the work in [17,22] discuss how to select and group dynamically generated Web pages in cache servers for performance improvement.

The work by Li et al. in [23] describe the deployment of NEC's CachePortal dynamic content caching technology on a database-driven Web site in a data center-based distribution infrastructure. The new system architecture has been experimentally evaluated to be able to accelerate the dynamic content delivery up to seven times. The work in [28] propose a *freshness-driven adaptive dynamic content caching*, which monitors the system status and adjusts caching policies to provide content freshness guarantees. This related work does not discuss adaptive database synchronization and load forwarding to maintain QoS as described in this paper.

10. Conclusion

Wide-area database replication and CDN technologies make data center an ideal choice to host and run database-driven Web applications that demand response time and reliability. We point out that such an architecture lacks capability to support many content sensitive Web-based applications, such as applications for financial markets, corporate cash management, supply chain and inventory management for large distributed retail chain stores, and decision making systems. We propose a new system architecture to coordinate and synchronize data centers to provide QoS for application precision and transaction response time. The proposed solution is effective and efficient as verified by the extensive evaluations through experiments. We also compare our algorithms with two versions of round robin scheduling algorithms. The experimental results validate our observation that our algorithm that combining adaptive scheduling and load forwarding and redirection is the most suitable for dynamic environment and various load distribution scenario.

Acknowledgement

The authors would like to acknowledge the contribution of Ullas Nambiar in the early version of this work.

References

- [1] A. Heddaya, S. Mirdad, D. Yates, Diffusion-based caching: WebWave, in: Proceedings of the 1997 NLANR Web Caching Workshop, 1997.
- [2] M.R. Korupolu, M. Dahlin, Coordinated placement and replacement for large-scale distributed caches, in: Proceedings of the 1999 IEEE Workshop on Internet Applications, 1999.
- [3] A. Heddaya, S. Mirdad, WebWave: globally load balanced fully distributed caching of hot published documents, in: Proceedings of the 1997 IEEE International Conference on Distributed Computing and Systems, 1997.
- [4] B. Adelberg, H. Garcia-Molina, B. Kao, Applying update streams in a soft real-time database system, in: Proceedings of ACM SIGMOD Conference, San Jose, California, June 1995, pp. 245–256.
- [5] B. Adelberg, B. Kao, H. Garcia-Molina, Database support for efficiently maintaining derived data, in: Proceedings of 5th International Conference on Extending Database Technology (EDBT'96), Avignon, France, March 1996.
- [6] Akamai Technology. Information available at <<http://www.akamai.com/html/sv/code.html>>.
- [7] M. Altinel, C. Bornhoevd, S. Krishnamurthy, C. Mohan, H. Pirahesh, B. Reinwald, Cache tables: paving the way for an adaptive database cache, in: Proceedings of 29th VLDB conference, 2003.
- [8] C. Bornhövd, M. Altinel, S. Krishnamurthy, C. Mohan, H. Pirahesh, B. Reinwald, Dbcache: middle-tier database caching for highly scalable e-business architectures, in: Proceedings of ACM 2002 SIGMOD Conference, 2003.
- [9] R.L. Carter, M.E. Crovella, On the network impact of dynamic server selection, *Computer Networks* 31 (23–24) (1999) 2529–2558.
- [10] C. Chekuri, R. Motwani, Precedence constrained scheduling to minimize weighted completion time, *Discrete Applied Mathematics* 98 (1–2) (1999).
- [11] Y. Chen, R.H. Katz, J.D. Kubiawicz, Dynamic replica placement for scalable content delivery, in: Peer-to-Peer Systems: First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 2002, pp. 306–318.
- [12] J. Cho, H. Garcia-Molina, Synchronizing a database to improve freshness, in: Proceedings of ACM SIGMOD Conference, Dallas, Texas, May 2000.

- [13] Oracle Corp. Available from <<http://www.oracle.com/>>.
- [14] M.J. Franklin, M.J. Carey, M. Livny, Transactional client-server cache consistency: alternatives and performance, *ACM Transactions on Database Systems* 22 (3) (1997) 315–363.
- [15] M.J. Franklin, B.T. Jónsson, D. Kossmann, Performance tradeoffs for client-server query processing, in: *Proceedings of SIGMOD*, 1996, pp. 149–160.
- [16] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, P. Keleher, Adaptive replication in peer-to-peer systems, in: *Proceedings of the 24th IEEE International Conference on Distributed Computing Systems*, Tokyo, Japan, 2004.
- [17] W.-P. Hsiung, W.-S. Li, K. Selçuk Candan, D. Agrawal, Multi-tiered cache management for e-commerce Web sites, in: *Proceedings of Second International Workshop on Cooperative Internet Computing (CIC 2002)*, Hongkong, China, August 2002.
- [18] Y. Ioannidis, The history of histograms, in: *Proceedings of 29th VLDB conference*, Berlin, Germany, 2003.
- [19] D. Kossmann, M.J. Franklin, G. Drasch, W. Ag, Cache investment: integrating query optimization and distributed data placement, *ACM Transactions on Database Systems* 25 (4) (2000) 517–558.
- [20] A. Labrinidis, N. Roussopoulos, Update propagation strategies for improving the quality of data on the Web, in: *Proceedings of 27th VLDB Conference*, Roma, Italy, 2001.
- [21] E.L. Lawler, Sequencing jobs to minimize total weighted completion time, *Annals of Discrete Mathematics* 2 (1978) 75–90.
- [22] W.-S. Li, W.-P. Hsiung, D.V. Kalashnikov, R. Sion, O. Po, D. Agrawal, K. Selçuk Candan, Issues and evaluations of caching solutions for Web application acceleration, in: *Proceedings of the 28th Very Large Data Bases Conference*, Hongkong, China, August 2002.
- [23] W.-S. Li, O. Po, W.-P. Hsiung, K. Selçuk Candan, D. Agrawal, Acceleration of data center-hosted distributed database-driven Web applications, in: *Proceedings of Second International Workshop on Cooperative Internet Computing (CIC 2002)*, Hongkong, China, August 2002.
- [24] Q. Luo, S. Krishnamurthy, C. Mohan, H. Pirahesh, H. Woo, B. Lindsay, J.F. Naughton, Middle-tier database caching for e-business, in: *Proceedings of SIGMOD*, 2002.
- [25] C. Olston, B.T. Loo, J. Widom, Adaptive precision setting for cached approximate values, in: *Proceedings of ACM SIGMOD Conference*, Santa Barbara, California, May 2001.
- [26] E. Pacitti, E. Simon, R.N. Melo, Improving data freshness in lazy master schemes, in: *Proceedings of 18th International Conference on Distributed Computing Systems*, Amsterdam, The Netherlands, May 1998.
- [27] R. Tewari, M. Dahlin, H.M. Vin, J.S. Kay, Design considerations for distributed caching on the Internet, in: *Proceedings of the 19th International Conference on Distributed Computing Systems*, 1999.
- [28] W.-S. Li, O. Po, W.-P. Hsiung, K. Selçuk Candan, D. Agrawal, Engineering and hosting adaptive freshness-sensitive Web applications on data centers, in: *The Proceedings of the 12th WWW Conference*, Budapest, Hungary, May 2003.
- [29] O. Wolfson, S. Jajodia, Y. Huang, An adaptive data replication algorithm, *ACM Transactions on Database Systems* 22 (2) (1997) 255–314.
- [30] H. Yu, A. Vahdat, Design and evaluation of a continuous consistency model for replicated services, in: *Proceedings of 4th Symposium on Operating Systems Design and Implementation*, October 2000.



Wen-Syan Li is a Senior Research Staff Member at NEC Laboratories America, Inc. He received his Ph.D. in Computer Science from Northwestern University in December 1995. He also holds an MBA degree. His main research interests include content delivery network, multimedia/hypermedia/document databases, WWW, e-commerce, and information retrieval. Wen-Syan is the recipient of the first NEC USA Achievement Award for his contributions in technology innovation.



Kemal Altintas is a Ph.D. student in Information and Computer Science Department, University of California, Irvine. He received his BS and MS degrees in Computer Science from Bilkent University, Turkey. His research interests include topics on data and information management systems, information retrieval and extraction from text and speech and natural language processing.



Murat Kantarcioğlu is a Ph.D. candidate at Purdue University. He has a Master's degree in Computer Science from Purdue University and a Bachelor's degree in Computer Engineering from Middle East Technical University, Ankara Turkey. His research interests include data mining, database security and information security. He is a student member of ACM.