A Federated Approach for Increasing the Timely Throughput of Real-Time Data Services

Yan Zhou and Kyoung-Don Kang
Department of Computer Science
State University of New York at Binghamton
{*yzhou,kang*}*@cs.binghamton.edu*

Abstract

As the demand for real-time data services (e.g., e-commerce or online auctions) increases, it is desired for a real-time database to increase the timely throughput—the amount of data processed in a timely manner. As the timely throughput of a centralized real-time database is limited, it is desired to federate a set of real-time databases to increases the timely throughput. However, related work on distributed real-time databases is scarce. Most existing approaches are highly complex, incurring non-trivial overheads. Neither are they implemented in a real database system. To address the problem, we design a new system architecture for federated real-time data services and develop efficient approaches for load sharing among a set of clustered databases. To support the desired data service delay even in the presence of dynamic workloads, each individual database employs a single-input single-output (SISO) feedback admission control scheme. Based on the admission control signals collected from the individual databases, cluster-wide load sharing is performed to enhance the total timely throughput by fully utilizing the federated databases, while avoiding to overload them. We have implemented and evaluated the performance of our approach by extending the Oracle Berkeley DB. Our system significantly enhances the timely data throughput compared to a single centralized system, while effectively dealing with emulated partial unavailability of a set of federated databases.

1 Introduction

Soft real-time data services are needed in various applications such as e-commerce, online auction, and traffic monitoring to process data service requests in a timely manner. In these applications, it is essential to process transactions and queries (i.e., read-only transactions) in a timely manner using fresh data that capture the current market or traffic status. The effectiveness of real-time data services depends not only on the logical results of data service requests but also on the time within which the results are produced. As the demands for real-time data services grow beyond the processing capacity of stand-alone database system, either the service delay increases or the system throughput drops considerably due to overloads. Although real-time data management has been studied extensively, most existing work focuses on a centralized single real-time database (RTDB) system, which limits the timely throughput of real-time data services [17]. To support the timeliness, a single RTDB may reject an excessive number of data service requests under overload. As a result, the timely throughput—the total amount of data processed within a specified average response time bound such as 1s—may decrease significantly.

A possible approach to addressing this problem is to cluster a set of RTDBs. However, this seemingly simple approach creates complex issues. In a distributed database, a transaction can be processed in any database node as data are replicated. At the same time, the distributed database should support 1-copy serializability that requires the result of executing distributed transactions is equal to the result of executing the transactions in some sequence using single (non-replicated) data instances. To support 1-copy serializability, a distributed database has to employ a computationally expensive mechanism such as the two phase commit (2PC) protocol and complex data consistency model [16]. Although previous work has been done to reduce the overhead for executing distributed real-time transactions [17, 24], most existing work focuses on developing complex real-time concurrency control protocols, potentially incurring large overheads. Also, none of them has actually been implemented in a real database system. Wei et al. [22] have designed a fully replicated distributed RTDB, in which the system-wide load balancer distributes incoming data service requests to the database nodes in a local area network (LAN) and each node applies feedback-based admission control. However, their work is not implemented in a real database either.

To address the problem, we design a new RTDB architecture to leverage a federated RTDBs clustered together via network switch (e.g., an Ethernet or Infiniband switch) to process transactions and queries in a timely, coordinated fashion. More specifically, we extend the distributed RTDB model developed by Wei et al. [22] to enhance the efficiency of real-time data services via *integrated cluster-wide load balancing and feedback-based admission control at each individual node*. Although load balancing has extensively been studied, surprisingly little work has been done on load sharing in a federated RTDBs [17, 22]. In fact, it is challenging to systematically distribute data service requests to a clustered RTDB nodes. Coarse-grained load distribution policies could easily lead to uneven load distribution. For example, if only the number of data service requests are balanced among the nodes, some nodes may have to process bigger transactions that access larger amounts of data. As a result, they can be overloaded. To support the desired data service delay bound such as 1s, they may have to unnecessarily reject incoming data service requests even if the other nodes are underutilized, substantially decreasing the total timely throughput of the federated RTDBs. In addition, the workload distribution algorithm may considerably affect the data locality of the transaction/query processing in the clustered databases. For this reason, we take *data access patterns* into account during the design of load distribution algorithm would further benefit the system timely throughput. Specifically, we design and develop a new clustered QoS-aware database system, called *Chronos-C* that extends Chronos [8]−a centralized QoS-aware database−to enhance the timely throughput of online data services such as e-commerce or online auctions. The key contributions of this paper are to (1) design a new system architecture for clustered real-time data services, (2) develop a cost-effective load balancing scheme specific to federated RTDBs, and (3) thoroughly evaluate the performance of our approaches by extending Oracle Berkeley DB [2]−a popular open source database.

Chronos-C is a logical unification of independent database servers connected by a high-speed network switch. We extend the *single-master-multiple-slave architecture* [3] for real-time data services: transactions are processed by a designated master database that replicates the result of database updates, if any, to every slave database to support data consistency based on the notion of 1-copy serializability [16]. In this way, we ensure that all the clustered databases share a consistent view of the data. As most existing databases support an efficient data replication scheme in the master-slave mode, our approach supports 1-copy serializability with less complexity and overhead than a distributed database counterpart does. In our clustered system, slave databases only process queries. They share no resources with each other and, therefore, run independently. At the same time, to support data freshness, temporal data such as stock prices are simultaneously multicast, i.e., replicated, to the master and the slave nodes. Thus, a query can be processed by any database node for load sharing and timely throughput enhancement. It is known that most online data service requests are queries and transactions are only a small fraction of the requests [19, 3]. Thus, a single-master-multiple-slave architecture is suitable for online real-time data services. Given a large number of data service requests, our approach can significantly improve the timely throughput with little overhead compared to a single, centralized real-time database system model mainly considered in the RTDB research [17, 14].[1] In Chronos-C, clustered databases share query processing workloads using fresh data to reduce the data service delay, while enhancing the timely throughput. Moreover, a slave database unavailability due to, for example, a physical node reallocation to different applications in a utility computing environment or an unexpected slave node failure simply reduces the timely throughput of the clustered database system without compromising data consistency. Since slaves only process queries, a removal of a slave node simply eliminates a database replica.

In this paper, we develop database-specific load sharing scheme. Each database node applies feedback-based admission control to incoming real-time data service requests to closely support the average data service delay. As the amount of data to process increases, the timeliness decreases and vice versa. Based on this observation, we design a single-input single-output (SISO) admission controller, similar to [9]. The *SISO controller in each node dynamically adjusts the database load bound* expressed as the amount of data to process, if necessary, to closely support a specified average response time, e.g., 1s, in the individual node. An incoming data service request is admitted, if the load bound is not exceeded by adding the estimated amount of data to be accessed by the incoming request. The cluster-wide load balancer periodically collects the load bound from each RTDB node in the cluster. In proportion to the load bounds received from the RTDB nodes, the load balancer adjusts the workload distribution among the nodes for the next sampling period. At the same time, it forwards data service requests accessing similar data to the same nodes such that cached data can be used to process data service requests. Note that we do not choose to develop a centralized multiple-input, multiple-output (MIMO) controller running in the load balancer using the performance data collected from the slaves. If the slave servers are mod-

[1] In this paper, we assume that the master node is not a bottleneck, since it only processes transactions, e.g., sell/buy transactions for stock trading, and it can be configured to run on a node with abundant resources. Thus, the master node accepts and processes all incoming transactions with no admission control.

eled in an inter-dependent manner in a MIMO model, an addition (join) and removal (leave/failure) of a database node may adversely affect the accuracy of overall system model. By running an independent SISO control loop in each node, we *avoid any potential impact of partial unavailability of the slaves on the model accuracy*. The general database operation and data consistency is not affected in our single-master-multi-slave architecture. As temporal data updates as well as the results of the transactions committed at the master are efficiently replicated to every slave, a *failover of the master is straightforward and efficient*; that is, any slave node can take over the role of the master when the master fails. Furthermore, our load balancer is *stateless*. For load balancing, it only needs the load bounds computed by the slave nodes for the next sampling period. Thus, any node can take the load balancing role, if the load balancer fails. According to [3], little work has been done to support load balancing and replication, while considering the failover of the load balancer. To the best of our knowledge, the work presented in this paper is the first to support *immediate* failover of the master, slaves, and load balancer, while supporting the desired delay for real-time data services. This is a desirable feature for real-time data services in that recovering a replicated database often takes hours or even days [3].

We compare our approach to to two baseline approaches also implemented in Berkeley DB. To evaluate the effectiveness of our approaches, we increase the workload and the number of slave database nodes from 1 to 5. Our approach closely supports the desired average and transient delay for data services. In contrast, the data service delays of the baselines largely exceed the desired set-point, e.g., 1s. Further, our approach show the largest timely throughput. Also, our timely throughput increases faster than the baselines do for the increasing number of slaves. The timely throughput of the closed-loop approach up to 3.37 (and 2.28) times the timely throughput of the baseline approach that shows the lowest (and second lowest) timely throughput. In addition, we evaluate the performance of the tested approaches when a subset of the slave nodes are intentionally removed from the cluster. In this scenario, our approaches significantly outperform the tested baselines. They process approximately $10 - 16$ million more data items in a timely fashion than the tested baselines do in a 10 minute experiment. Our approach is lightweight. Feedback-based admission control running at each slave node only consumes approximately 1% CPU utilization. Our basic load balancing scheme that distributes workloads to the clustered databases consumes approximately 5% CPU utilization in the load balancer when 7500 client threads concurrently submit data service requests to the load balancer that distribute workloads among the clustered databases.

The remainder of this paper is organized as follows. The overall structure of our clustered database system and the overview of our approach are described in §2. A description of feedback control design is given in §3. Our load balancing scheme is discussed in §4. Performance evaluation results are described in §5. §6 discusses related work. Finally, §7 concludes the paper and discusses future work.

2 Database Model and System Overview

In this section, our database model and overall system architecture for federated real-time data services, a high-level description of our closed-loop approach to supporting the desired data service delay in a cluster of databases, and an example real-time data service level agreement in terms of the average and transient delay are discussed.

In this paper, we consider a federated database system that consists of a group of databases connected by a network switch. Each database node hosts a set of temporal data objects and non-temporal data objects. Temporal data updated periodically by dedicated update transactions to maintain the temporal consistency between the real-world state and data in the database [17]. In contrast, non-temporal data values do not change dynamically with time. For instance, in stock trading applications, temporal data include stock prices and volumes. Non-temporal data include stock IDs and company names.

In our approach, only the master node executes transactions. When a user transaction commits, it replicates the result to the slaves to support 1-copy serializability [16]. In our testbed, this is implemented using the data replication library provided by Berkeley DB. For concurrency control, we apply the two phase locking (2PL) algorithm in each local database node. Note that no distributed lock management or data conflict resolution is needed because transactions are only executed in the master. Transactions are not associated with deadlines since most online trade transactions do not have explicit deadlines. Also, they are scheduled in a FCFS manner as common in online data services. However, a transaction is required to be processed within a specified response time bound. Otherwise, online users may simply leave. As 2PL and FCFS are supported by most existing database systems, our model is easy to deploy.

Figure 1 shows the architecture of Chronos-C. The databases are clustered in a single-master-multiple-slave fashion. The master server processes all transactions, while the slaves process queries as discussed before. When a transaction commits, the master node replicates the database status updates caused by the transaction to the slave databases to maintain the data consistency. The user service requests, i.e., transactions and queries, are first sent to the load balancer that forwards transactions to the master node, while distributing queries among the slave nodes. Each slave database process queries and periodically reports

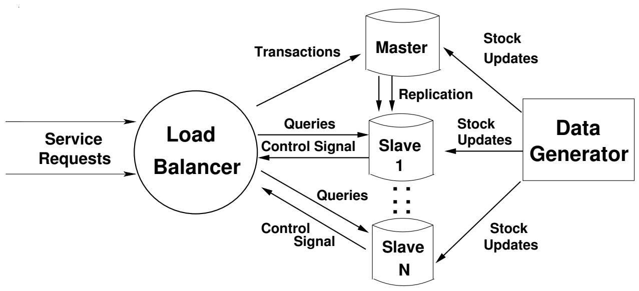

Figure 1. Federated RTDB Architecture

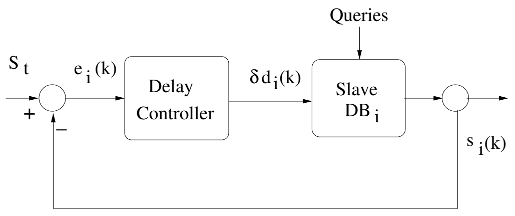

Figure 2. Delay Control at Slave i

its load bound computed in its feedback control loop to support the desired service delay to the load balancer at each sampling point. Based on the feedback control signals received from the slave nodes, the load balancer makes load sharing decisions, which determines the fractions of incoming queries to be distributed to the slave database nodes.

We consider periodic temporal data updates, since periodic updates are commonly used in RTDBs to support the temporal data consistency [17, 23]. In our testbed, the update period of each temporal data is in a range of $[0.5s, 5s]$ to mimic frequent updates of stock prices, sensor data, etc. In our federated RTDB architecture, the network switch receiving temporal data updates from the real world, e.g., a stock market or roadside sensors for traffic monitoring, is configured to periodically multicast incoming temporal data to the slaves. An alternative approach is updating temporal data in one node and sharing them with the other nodes. In this paper, we take the first approach, because the network bandwidth for periodic temporal data updates can be preallocated. In the second approach, however, the network may become congested due to aperiodic, bursty accesses of shared temporal data across the network by user requests for online data services, which often arrive in a bursty manner. Upon receiving new temporal data, each database node immediately runs its dedicated threads for periodic updates. Admission control is only applied to user requests, if necessary, to closely support the desired delay for real-time data services by avoiding overload. In our approach, transactions are processed by the master node using fresh data. A user query can be processed in any slave node using the locally available fresh temporal data and non-temporal data whose consistency with the master is guaranteed via 1-copy serializability.

3 Feedback Control Design

In each slave node, we apply feedback control techniques [7] to support the desired data service delay even in the presence of dynamic workloads. For feedback control, the k^{th} (≥ 1) sampling period is the time interval $[(k-1)P, kP)$ and the k^{th} sampling point is equal to time kP where P is the sampling period. In this paper, we set P = $1s$. As hundreds of queries finish in 1s in each slave server, performance measurement for $P = 1s$ is reliable. All the tested approaches use the same sampling period for performance evaluation in Section 5. By applying feedback control in each slave node, we aim to support timely data services. In this paper, we consider an example service level agreement (SLA): SLA = $\{S_t = 1s$, $S_v \leq 1.1s$, T_v = $10s$ $\}$. The average service delay is desired to be shorter than or equal to $S_t = 1s$. An overshoot S_v, if any, is a transient service delay longer than S_t. In this paper, it is desired that $S_v \leq 1.1s$. Also, an overshoot, if any, is desired to reduce to be equal to or less than S_t within the settling time T_v = $10s$.

The delay for servicing a request, s_i, is the sum of the TCP connection delay, queuing delay, and processing delay inside the database. Let us assume that $N \geq 1$ RTDB nodes are clustered together. At the k^{th} sampling point, the feedback controller running in each node shown in Figure 2 computes the service delay $s_i(k) = \sum_{j=1}^{n(k)} s_j/n(k)$ and delay error $e_i(k) = S_t - s_i(k)$ where $n_i(k)$ denotes the number of queries finished in node i during the k^{th} sampling period. Based on $e_i(k)$, the feedback controller in node i ($1 \leq i \leq N$) computes the control signal, i.e., the required load bound adjustment $\delta d_i(k)$, at the the k^{th} sampling point. Also, it computes the backlog bound for the next sampling period: $d_i(k+1) = d_i(k) + \delta d_i(k)$. Thus, the backlog bound is decreased (i.e., $\delta d_i(k) < 0$) to admit fewer queries, if the service delay is longer than S_t specified in the SLA or vice versa.

During the $(k+1)^{th}$ sampling period, node i parses an incoming data service request to estimate how many data items the request will access by leveraging the database schema and semantics of queries. (For more details about parsing, refer to [9].) If the sum of the estimated number of data items to be accessed by the request and the total number of the data to be accessed by the user requests already in node i does not exceed $d_i(k+1)$, node i admits the request. Otherwise, the request is rejected. To enhance the total timely throughput of the federated RTDBs, at the k^{th} sampling point, the load balancer uses the backlog bounds collected from the slave nodes for load balancing in the

$(k+1)^{th}$ sampling period.

3.1 Admission Control in A Slave Database Node

In each slave database node, we model the database system dynamics in terms of the relation between the service delay and the database backlog, i.e., the amount of data for the database to process, in a SISO manner. By using multiple, independent SISO controllers rather than a single, centralized controller for load balancing, we aim to enhance the robustness of real-time data services as discussed before. As the database backlog increases, the service delay increases and vice versa. Based on this observation, we model the RTDB behavior via the following four-step procedure.

1. System Modeling. We aim to construct a system model whose input is the database backlog and the measured output is the service delay. We derive a RTDB model in the discrete time domain using the ARX (Auto Regressive eXternal) model [7, 15]. Specifically, to model the relation between the database backlog and delay, we model the data service delay at the k^{th} sampling point via the service delays and database backlogs measured at the previous p sampling points. We express the relation in slave node i as a difference equation in the discrete time domain:

$$s_i(k) = \sum_{j=1}^{p} \{a_j s(k-j) + b_j d(k-j)\} \tag{1}$$

where p (≥ 1) is the system order [15]. $s(k-j)$ and $d(k-j)$ are the service delay and backlog measured during the time interval of $[(k-j)P, (k-j+1)P)$. Using this difference equation, we model database dynamics by considering individual queries that potentially access different amounts of data.

2. System Identification. The unknown model parameters a_j's and b_j's in Eq 1 are derived via system identification (SYSID) [15] to construct the system model. The objective of our SYSID is to minimize the sum of the squared errors of data service delay estimations based on database backlogs. In our SYSID procedure, we use the load balancer, the master node, and only one slave node. Since the slave nodes are homogeneous in terms of system capacity in our testbed, this approach saves the effort for conducting SYSID individually for each slave. If a federated database cluster consists of heterogeneous groups of nodes, SYSID is needed for only one server in each group of homogeneous nodes. This modeling approach is relatively simple and robust to changes in node availability compared to a MIMO approach tied to a specific cluster configuration.

For SYSID, 1500 client threads concurrently send data service requests to the load balancer for one hour. The load balancer forwards transactions to the master node and queries to the slave node. The master node replicates all the database updates caused by committed transactions to the slave node to support 1-copy serializability. Each client thread sends a service request and waits for the response from the database server. After receiving the transaction or query result, it waits for an inter-request time randomly selected in a range before sending the next data service request. A data service request accesses 60-100 data items. As SYSID aims to identify the behavior of the controlled database system, the master and slave nodes accepts all incoming data service requests without applying admission control.

For SYSID, we choose the inter-request time range [1s, 3.5s], since the service delay shows a near linear pattern in this area. In this way, we can model high performance real-time data services dealing with widely varying workloads. Our control modeling and tuning is valid in this operating range. Beyond the operating range, a feedback controller in each node may or may not support the desired performance [7]. By employing multiple slave databases, we can support the desired data service delay for a considerably extended operating range as long as the load distributed to each individual node does not largely exceed the operating range.

3. Model Evaluation. We use the R^2 metric computed by Eq 2 to analyze the accuracy of SYSID [15]:

$$R^2 = \frac{variance(service\ delay\ prediction\ error)}{variance(measured\ service\ delay)} \tag{2}$$

A control model is acceptable, if its $R^2 \geq 0.8$. We have performed SYSID for the first order to fourth order system models. We reject the first order model due to its poor R^2 value. We choose the second order model since its $R^2 = 0.872$:

$$s(k) = -0.0228s(k-1) - 0.1371s(k-2) + 0.0179d(k-1) + 0.0084d(k-2). \tag{3}$$

The third and fourth order model show slightly better R^2 value compared to the second order one. However, we choose the second model, since a higher order increases the complexity of the system model.

4. Controller Design and Tuning. To design the service delay controller, we derive the transfer function of the open-loop RTDB that models the relation between the backlog and service delay. Especially, we take the z-transform [7] of Eq 3 to algebraically manipulate the equation in the frequency domain rather than solving partial differential equations in the time domain. From this, we get the following transfer function that shows the relation between the service delay and backlog:

$$P_i(z) = \frac{S_i(z)}{D_i(z)} = \frac{0.0179z + 0.0084}{z^2 + 0.0228z + 0.1371} \tag{4}$$

where $S_i(z)$ is the z-transform of $s_i(k)$ and $D_i(z)$ is the z-transform of $d_i(k)$ in Eq 3.

To closely support the SLA, we apply an efficient PI (proportional and integral) control law, which combines the advantages of integral control (zero steady-state error) with that of proportional control (increasing the speed of the transient response) [7]. We do not use a D (derivative) controller sensitive to noise such as bursty arrivals of data service requests and data conflicts. At the k^{th} sampling point, the PI controller in node i computes the control signal $\delta d_i(k)$, i.e., the database backlog adjustment required to support S_t:

$$\delta d_i(k) = \delta d_i(k-1) + K_P[(K_I+1)e_i(k) - e_i(k-1)] \quad (5)$$

where the error $e_i(k) = S_t - s_i(k)$ at the k^{th} sampling point as shown in Figure 2. The z-transform of Eq 5, is:

$$F_i(z) = \frac{\Delta D_i(z)}{E_i(z)} = \frac{K_P(K_I+1)[z - 1/(K_I+1)]}{z-1} \quad (6)$$

where $\Delta D(z)$ and $E(z)$ are the z-transform of $\delta d(k)$ and $e(k)$, respectively. Using Eq 4 and Eq 5, one can derive a transfer function for the closed-loop system in Figure 2 and tune the control gains, i.e., K_P and K_I, via well established control theoretic techniques [7, 15]. We have derived the closed-loop transfer function and tuned K_P and K_I to support the stability of the closed loop system and SLA considered in this paper. Details of these standard procedures are omitted due to space limitations.

4 Load Balancing among Slave Databases

In this section, two approaches for load balancing among slave RTDBs are described.

FC-Prop: Proportional Load Balancing Our first approach, called FC-Prop, distributes incoming queries in proportion to the backlog bounds computed by the individual slave nodes via feedback control. At the k^{th} sampling point, the load balance computes the fraction of the incoming query workload to be assigned to slave node i during the $(k+1)^{th}$ sampling period:

$$f_i(k+1) = d_i(k+1) / \sum_{i=1}^{n} d_i(k+1) \quad (7)$$

where $\sum_{i=1}^{n} f_i(k+1) = 1$. Note that FC-Prop distributes the load in a *fine-grained* manner, considering the potential *heterogeneity of data service requests*. The database backlog bound computed in each individual slave indicates the number of data accesses (rather than the number of data request) allowed to meet a specified average delay bound for data services in the next sampling period based on the observation that different data service requests may access different numbers of data. Also, FC-Prop efficiently handles *potential node heterogeneity*. For example, assume both nodes A and B closely support a specified average delay bound in a sampling period. However, node A can process 10,000 data/s while B process 30,000 data/s to support the desired average delay due to the difference in terms of their hardware resources. A gross-grained approach may assign the same number of data service requests to nodes A and B in the next sampling period, overloading node A. Since Eq 7 is computed based on the fine-grained backlog bounds computed by all the individual slave nodes active at the k^{th} sampling point, FC-Prop can effectively deal with the potential heterogeneity of data service requests and slave nodes, enhancing the total timely throughput of the federated RTDBs.

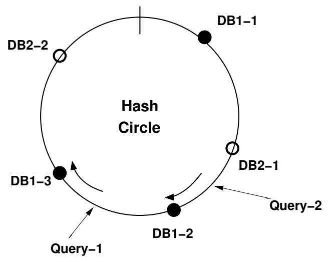

Figure 3. Consistent Hashing Example

FC-Hash: Load Balancing Considering Data Access Patterns. We further enhance the effectiveness of load sharing for federated real-time data services by taking advantage of the locality in transaction processing. This new approach is called FC-Hash, in which we extend consistent hashing [10] and integrate it with FC-Prop. Figure 3 shows an example of how our extended consistent hashing distributes incoming queries between two slave servers, DB1 and DB2 in a cluster. In consistent hashing, an output of the hash function is mapped to a point on a circle; therefore, the largest hash value wraps around to the smallest hash value as shown in Figure 3. For hashing, we use MD5 hash function. We use the least significant 32-bits of the 128-bit MD5 hash value and map it to one of the $H = 2^{32} - 1$ points on a hash circle. At the k^{th} sampling point, FC-Hash computes the number of hash points assigned on the hash circle to active slave node i that indicates the number of imaginary database nodes run by slave node i:

$$H_i(k+1) = f_i(k+1)H \quad (8)$$

Thus, more imaginary nodes will be assigned to physical slave node i on the hash circle, if $d_i(k+1)$ is larger than that of the other slaves or vice versa. If a query arrives in the $(k+1)^{th}$ sampling period, it is hashed to an initial position on the hash circle. Starting from the initial location, the ring is searched clockwise to find the first available hash point, i.e., imaginary server. For example, suppose Query 1 is hashed between DB1-2 and DB1-3 in Figure 3, which are imaginary instances of node 1 on the hash circle. FC-Hash searches clockwise around the circle until it finds an imaginary server DB1-3. Therefore, the load balancer sends this query to physical DB1. Notably, H_i varies as the data service delay and corresponding load bound provided by node i change dynamically from a sampling point to another.

In addition to integrating feedback-based load balancing with consistent hashing, we leverage data access patterns to improve the performance by exploiting database caching. Specifically, data identifiers, e.g., online stock price or product IDs, are used as the input to consistent hashing. Thus, queries accessing the same data will be hashed to the same imaginary server and same physical database node accordingly. Since the recently accessed data is cached for direct access, the query processing delay can be reduced by using our extended approach for consistent hashing, which effectively integrates load balancing, feedback control, consistent hashing, and database caching altogether to enhance the timeliness of data services. Also, note that load sharing by FC-Prop or FC-Hash only directs the way the load balancer distributes workloads. Thus, it does not affect the system model based on the relation between the service delay and backlog in each individual slave described in Section 3.

5 Performance Evaluation

In this section, the federated real-time data service architecture and load balancing schemes presented in this paper are implemented and evaluated.

5.1 Experimental Settings

For performance evaluation, we use a stock trading testbed that provides four types of transactions: view-stock, view-portfolio, purchase, and sale for seven tables [9], similar to the TPC-W benchmark [19]. Different from TPC-W, our testbed also supports periodic updates of 3,000 stock prices (i.e., temporal data) for real-time data services. All the machines used in the experiments have the dual core 1.6GHz CPU, 1GB memory and the 2.6.23 Linux kernel. One database server runs on one dedicated physical machine and each server has 500MB database cache. Database nodes are connected by a 1Gbps Ethernet switch. One experiment runs for 600s. Each performance data is the average of 10 runs with 90% confidence intervals. We show the performance results observed between $100s$ and $600s$ to exclude the database initialization phase, involving initial housekeeping chores, e.g., database schema and data structure initialization.

For 90% of time, a client thread issues a query (read-only transaction) about stock prices or portfolio browsing, similar to real world e-commerce workloads [3, 19]. For the remaining 10% of time, a client sends a purchase or sale transaction. We model bursty workloads to observe the performance of tested approaches when the workload changes dramatically. At the beginning of one experiment, the inter-request time (IRT) is randomly distributed in [3.5s, 4s]. In all the experiments presented in this paper, at 200s, the range of the IRT is suddenly reduced to [1s, 1.5s] to model bursty workload changes and stays in the new range until the end of each experiment. The sudden decrease of the IRT at 200s increases the workload by approximately $2.33 - 4$ times. Note that we only show the performance of the slave database nodes. The desired 1s delay is always supported for transactions processed in the master node, since only 10% of the service requests are transactions in our experiments. Also, the master node is always up and running in our experiments.

For performance comparisons, we consider the following baseline approaches commonly used for load balancing in addition to FC-Prop and FC-Hash described in §4:

- **RR**: In this approach, service requests will be distributed to the database servers in a round robin manner without considering the sizes of individual data service requests measured in terms of the number of data accesses. Neither feedback control is applied to support the desired delay bound in each node.
- **Prop:** The load balancer distributes workloads in proportion to the inverse of the service delays measured in the slaves. Essentially, this is an ad hoc feedback control in that load distribution among the slave databases considers dynamically changing status of the slaves without applying formal control theoretic techniques.

Note that RR and Prop are built based on the same single-master-multi-slave architecture shown in Figure 1. Thus, they share the advantage of high availability with FC-Prop and FC-Hash even in the presence of RTDB node failures or reassignments to different applications.

In this paper, we perform two major groups of experiments. The first group of experiments evaluates the performance when the number of slave databases increases. The second group evaluates the performance when the number of slave databases decreases to mimic partial system unavailability. In the first group of experiments, we start with one slave to which 1500 client threads send service requests. We conduct 5 sets of experiments by increasing the number of slave databases from 1 to 5. In one experimental run, the number of slaves is fixed. Each time we increase the number of slaves by 1, we add 1500 client threads to stress the clustered databases. Thus, for the case of a cluster with 5 slaves, 7500 client threads submit data service requests to the cluster. In the second group of experiments, we start experiments with 1 master and 5 slaves to which 7500 clients simultaneously send service requests. To mimic partial cluster unavailability, we remove 1 or 2 slaves out of 5 slaves at 400s, while keeping the same number of clients.

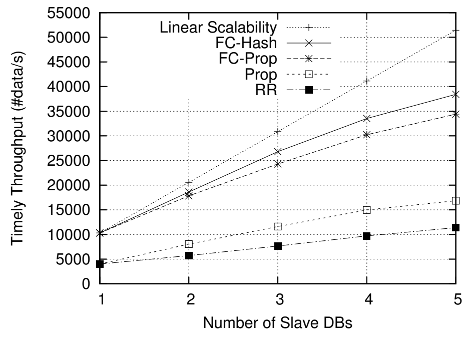

Figure 4. Average Timely Throughput

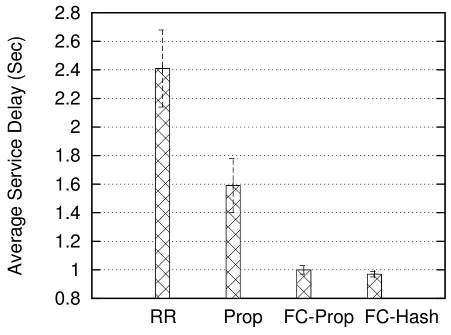

Figure 5. Average Delay (5 Slaves)

In this way, we support a failover fault model (assuming that there is no Byzantine failure). Hence, from 400s to the end of experiments, all service requests are processed by the remaining database servers. Although we have done extensive performance evaluation, due to space limitations, we only present a subset of key results next.

Performance for an Increasing Number of Slaves. Figure 4 shows the the timely throughput, i.e., the total number of data processed by timely queries that finish within the desired delay bound. In general, as the number of slaves increases from 1 to 5, the total and timely throughput increase for all the tested approaches. FC-Hash and FC-Prop consistency support higher timely throughput Notably, the throughput gap between our approaches and Prop/RR in Figure 4 increases as more slave databases are employed. In the 5 slave server case, the timely throughput of FC-Hash is 3.37 and 2.28 times the timely throughput of RR and Prop as shown in Figure 4. By applying formal control theoretic techniques to manage the database backlog, FC-Prop and FC-Hash avoid severe database overload. In addition, FC-Hash is aware of data access patterns and takes advantage of database caching to further improve the performance compared to FC-Prop. Its timely throughput in Figure 4 is higher than FC-Prop's by roughly 4000 date items/s in the 5 slave case. The linear line in Figure 4 indicates the ideal

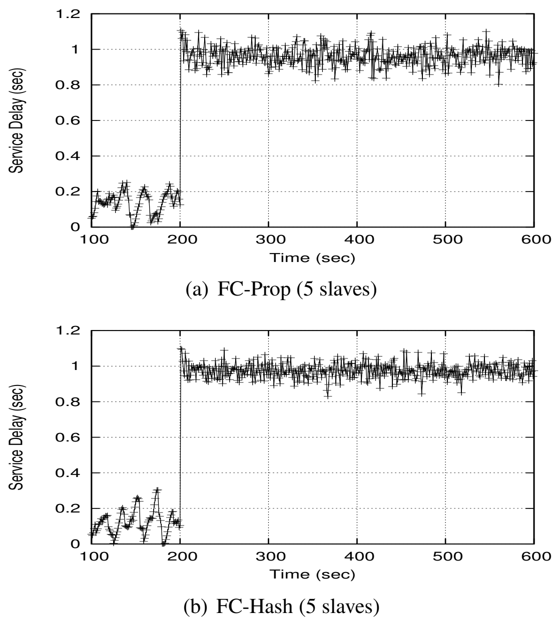

Figure 6. Transient Service Delay

linear increase of the timely throughput for the increasing number of slave databases. In the future, we will explore more efficient load balancing and RTDB clustering techniques to further reduce the gap between the timely throughput of our approaches and the ideal linear line.

Figure 5 shows the average of service delays observed for 10 runs across 5 slave nodes with 90% confidence interval bars. As shown in the figure, RR and Prop support the average service delay of 2.41 ± 0.27s and 1.59 ± 0.19s, respectively, violating the 1s average delay bound specified in the example SLA considered in this paper (§3). In contrast, FC-Prop and FC-Hash support 1 ± 0.03s and 0.97 ± 0.02s, closely supporting the 1s bound. Also, their confidence intervals are an order of magnitude smaller than that of RR and Prop. Due to the space limitation, we only discuss the 5 slave database case. Other cases with $1 - 4$ slave databases have shown similar results.

In addition, Figure 6 shows the transient service delay of FC-Hash and FC-Prop. As RR and Prop cannot support even the average service delay bound, $S_t = 1s$, we do not plot their transient delay. In Figure 6, we only show the transient service delay of one slave database, DB-1, because FC-Hash and FC-Prop closely support the desired average and transient delay in every slave. Also, the delay variations across the clustered database nodes are negligible in FC-Hash and FC-Prop. As shown in Figure 6, FC-Hash and FC-Prop closely support the desired 1s delay bound after the abrupt workload change at 200s, while cancelling transient delay overshoots. Generally, the transient delay of FC-Prop and FC-Hash in Figure 6 ranges between 0.8s and 1.1s.

They have closely supported the desired average/transient delay bound specified in the SLA for the $1-4$ slaves too. We omit the results due to space limitations. FC-Hash provides similar transient delays to FC-Prop, while supporting a higher total timely throughput than FC-Prop does as discussed before.

Performance for Removing Slaves. After the number of slaves is decreased from 5 to 4 by removing one node at 400s, the *timely* throughput of FC-Hash and FC-Prop is at least twice the timely throughput of RR and Prop both before and after the removal of a slave as summarized in Table 1. From Table 1, we observe that a large fraction of data accesses in RR and Prop are tardy. As a result, RR and Prop suffer the significantly lower timely throughput. Similar performance results are observed when two slave nodes are abruptly removed from the cluster at 400s. These results emphasize the importance of real-time data services via systematic performance management.

6 Related Work

Although feedback control has been applied to manage the performance of a stand-alone RTDB [1, 8, 9], little work has been done to manage the performance of clustered RTDBs. Wei et al. [22] developed a closed-loop scheme to provide QoS-aware data services in a distributed RTDB working in a local area network environment with full data replication. In their work, the local deadline miss ratio and utilization controllers apply admission control to incoming transactions. The global load balancer collects the performance data from the database nodes and balances the system-wide workload. However, their work is based on heuristics rather than formal control theory, providing no stability analysis. Further, it is not implemented and evaluated in a real database system.

A feedback-based CPU utilization control algorithm for distributed soft real-time applications is presented in [20, 6]. However, real-time data management issues such as database-specific load balancing based on the notion of database backlog, consistent hashing, and data freshness issues are not considered in their work. Another work [5] designed a MIMO-control-based load balancing algorithm for optimizing the system response time through memory pool allocations to multiple disk storages in one database server. They focus on the trade-off between the performance cost of transient imbalance and the cost of control actions such as adjusting memory pools. Their approach aims to optimize the performance of multiple disks in one database server, while our work focuses on timely data services by clustered RTDBs.

Distributed real-time data management issues have been studied [13, 18]. MIRROR [24] is a concurrency control protocol developed for distributed RTDBs where data are replicated. Efficient algorithms to maintain the consistency of replicas in a distributed RTDB have been studied in [21]. In this paper, we take a different system design choice. By clustering independent databases based on the master slave architecture, we develop a new clustered RTDB architecture to significantly reduce the complexity and overhead of supporting the consistency of temporal and non-temporal data based on the notion of 1-copy serializability. Also, we are not aware of any prior work on distributed RTDBs implemented and evaluated in a real database system. As the database architecture is one of many possible factors that affect the timely throughput of real-time data services, a further study for more efficient system and algorithm design remains an open issue.

Consistent hashing has been applied to balance the load in web applications and data storage systems [12, 4, 11]; however, most existing approaches do not consider real-time data service issues such as supporting the desired average and transient data service delay in the presence of dynamic workloads, while enhancing the timely throughput by considering specific needs for real-time data services.

7 Conclusions and Future Work

In this paper, we develop a new architecture for clustered real-time data services as well as a load balancing scheme to enhance the timely throughput of real-time data services. Also, we implement our approach and evaluate the performance in a real database system. We present a closed-loop load balancing algorithm where the load distribution is based on the feedbacks from the service delay controllers running on the individual slave database servers. To further enhance the timely throughput via database caching, we also exploit data access patterns for load distribution. In sum, our federated real-time database system shows the largest timely throughput increase compared to baseline approaches as more slave nodes are added. Further, partial unavailability of the clustered database system only reduces the timely throughput without introducing complex data consistency issues. The performance of our approach is implemented and thoroughly evaluated in a real-time data service testbed. The performance results show that our approaches support the desired data service delay even in the presence of dynamic workloads and abrupt unavailability of a subset of the slave databases, while significantly improving the timely throughput compared to the tested baselines. In the future, we will perform more extensive experiments to evaluate the availability of the federated real-time database system using more slave nodes as well as more node additions or removals. Also, we will explore more advanced architecture of clustered real-time databases, while investigating more effective load sharing scheme.

Table 1. Throughput before and after removing a slave RTDB node

Metrics/Approaches	RR	Prop	FC-Prop	FC-Hash
#timely accesses/s before a removal	10,309	18,437	37,990	39,032
#timely accesses/s after a removal	9,024	15,526	33,128	35,627
#total accesses/s before a removal	41,647	43,499	50,197	54,821
#total accesses/s after a removal	35,379	38,251	46,068	50,946
total timely throughput over 600s	5,928,400	10,480,000	21,821,600	22,738,200

References

[1] M. Amirijoo, J. Hansson, and S. H. Son. Specification and Management of QoS in Real-Time Databases Supporting Imprecise Computations. *IEEE Transactions on Computers*, 55(3):304–319, 2006.

[2] Oracle Berkeley DB Product Family. Available at http://www.oracle.com/database/berkeley-db/index.html.

[3] E. Cecchet, G. Candea, and A. Ailamaki. Middleware-based Database Replication: The Gaps Between Theory and Practice. In *ACM SIGMOD International Conference on Management of Data*, 2008.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's Highly Available Key-value Store. In *Symposium on Operating Systems Principles*, 2007.

[5] Y. Diao, J. Hellerstein, A.Storm, and M. Surendra. Incorporating Cost of Control into the Design of a Load Balancing Controller. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2004.

[6] Y. Fu, H. Wang, C. Lu, and R. Chandra. Distributed Utilization Control for Real-Time Clusters with Load Balancing. In *IEEE International Real-Time Systems Symposium*, 2006.

[7] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. A John Wiley and Sons, Inc., Publication, 2004.

[8] K. D. Kang, J. Oh, and S. H. Son. Chronos: Feedback Control of a Real Database System Performance. In *IEEE Real-Time Systems Symposium*, 2007.

[9] K. D. Kang, J. Oh, and Y. Zhou. Backlog Estimation and Management for Real-Time Data Services. In *Euromicro Conference on Real-Time Systems*, 2008.

[10] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *ACM Symposium on Theory of computing*, 1997.

[11] D. Karger and M. Ruhl. Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems. In *ACM Symposium on Parallelism in Algorithms and Architectures*, 2004.

[12] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web Caching with Consistent Hashing. In *International Conference on World Wide Web*, 1999.

[13] K.-W. Lam, V. C. S. Lee, K.-Y. Lam, and S.-L. Hung. Distributed Real-time Optimistic Concurrency Control Protocol. In *International Workshop on Parallel and Distributed Real-Time Systems*, 1996.

[14] K. Y. Lam and T. W. Kuo, editors. *Real-Time Database Systems*. Kluwer Academic Publishers, 2006.

[15] C. L. Phillips and H. T. Nagle. *Digital Control System Analysis and Design (3rd edition)*. Prentice Hall, 1995.

[16] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 3rd edition, 2003.

[17] K. Ramamritham, S. H. Son, and L. C. Dipippo. Real-time databases and data services. *Real-Time System Journal*, 28:179–215, November 2004.

[18] J. Stankovic and S. H. Son. An Architecture and Object Model for Distributed Object-Oriented Real-Time Databases. *Journal on Computer Systems Science and Engineering*, 14(4):251–259, July 1999.

[19] Transaction processing performance council. http://www.tpc.org/.

[20] X. Wang, D. Jia, C. Lu, and X. Koutsoukos. DEUCON: Decentralized End-to-End Utilization Control for Distributed Real-Time Systems. *IEEE Transactions on Parallel and Distributed Systems*, 18(7):996–1009, 2007.

[21] Y. Wei, A. A. Aslinger, S. H. Son, and J. A. Stankovic. ORDER: A Dynamic Replication Algorithm for Periodic Transactions in Distributed Real-Time Databases. In *International Conference on Real-Time and Embedded Computing Systems and Applications*, 2004.

[22] Y. Wei, S. H. Son, J. A. Stankovic, and K. D. Kang. QoS Management in Distributed Real-Time Databases. In *IEEE International Real-Time Systems Symposium*, Dec. 2003.

[23] M. Xiong and K. Ramamritham. Deriving Deadlines and Periods for Real-Time Update Transactions. *IEEE Transactions on Computers*, 53(5):567–583, 2004.

[24] M. Xiong, K. Ramamrithm, and J. Haritsa. MIRROR: A State-Conscious Concurrency Control Protocol for Replicated Real-Time Databases. In *IEEE Real-Time Technology and Applications Symposium*, 1999.