A Predictive-Reactive Method for Improving the Robustness of Real-Time Data Services

Jisu Oh and Kyoung-Don Kang *Member, IEEE*

Abstract—Supporting timely data services using fresh data in data-intensive real-time applications, such as e-commerce and transportation management, is desirable but challenging, since the workload may vary dynamically. To control the data service delay to be below the specified threshold, we develop a predictive as well as reactive method for database admission control. The predictive method derives the workload bound for admission control in a predictive manner, making no statistical or queuing-theoretic assumptions about workloads. Also, our reactive scheme based on formal feedback control theory continuously adjusts the database load bound to support the delay threshold. By adapting the load bound in a proactive fashion, we attempt to avoid severe overload conditions and excessive delays before they occur. Also, the feedback control scheme enhances the timeliness by compensating for potential prediction errors due to dynamic workloads. Hence, the predictive and reactive methods complement each other, enhancing the robustness of real-time data services as a whole. We implement the integrated approach and several baselines in an open source database. Compared to the tested open-loop, feedback-only, and statistical prediction + feedback baselines representing the state of the art, our integrated method significantly improves the average/transient delay and real-time data service throughput.

Index Terms—Real-Time Data Services, Predictive and Feedback Control of Data Service Delays, Implementation and Performance Evaluation

1 Introduction

Data-intensive real-time applications, including e-commerce, traffic control, and military applications, need to process queries and transactions in a timely fashion using fresh (temporally consistent) data that reflect the current market or traffic status [2], [3]. In e-commerce, for example, stock quote queries and trade transactions should be processed within the acceptable response time bound using up-to-date stock prices. If the service delay is longer than a few seconds, most e-commerce clients tend to leave [4]. Transaction processing based on stale data, such as outdated stock prices, may adversely affect decision making. Similarly, data service requests for transportation management should be processed in a timely manner using fresh data representing the current traffic status.

Most existing (non-real-time) databases are unaware of the timeliness and data freshness requirements. They only aim to improve the long-term average response time or throughput. As a result, the transient data service delay may often exceed the desired threshold value by a large magnitude, even if the long-term average delay is acceptable. To address the problem, we aim to support the timeliness of data services in the context of real-time databases (RTDBs) [2], [3]. Data stream management systems (DSMS), such as [5]–[8], aim to support fast processing of stream data on the fly. Although related, DSMS and RTDBs have different data and system models. Generally, RTDBs periodically update temporal data, e.g., stock prices or sensor data, to support the temporal consistency of data within their confidence intervals [2], [3]. RTDBs intend to process one-time user transactions in a timely fashion using temporally consistent data. On the other hand, DSMS [5]–[8] usually assume that data may arrive in a bursty manner and execute the related continuous queries upon data arrivals. Although these data and transaction/query models are not fixed, DSMS and RTDBs are designed to deal with different scenarios [9], [10]. Thus, overload management techniques developed for RTDBs may not be directly applicable to DSMS and vice versa.

Supporting the desired timeliness and data freshness is challenging, since real-time data service requests may arrive in a bursty manner, for example, due to the varying market or traffic status. Also, the degree of data/resource contention may change from time to time. Due to these challenges, it is very hard, if at all possible, to provide hard real-time guarantees. In this paper, we consider soft real-time applications such as online trades or target tracking, in which a large service delay may degrade the quality of service (QoS) but does not cause a catastrophic result. Our goal is to control the average/transient data service delay to be below the specified threshold even in the presence of dynamic workloads, while supporting the freshness of data.

Feedback control [11], [12] is effective to manage the RTDB performance [1], [13]–[15]. By continuously measuring the data service delay and adapting the workload in the closed-loop, the desired data service delay bound can be closely supported. However, feedback control only reacts to performance errors. A feedback controller

An early version of this paper [1] was presented at the 20th IEEE Euromicro Conference on Real-Time Systems.

• *Jisu Oh is with SAMSUNG Electronics Co. Ltd, Korea. Kyoung-Don Kang (Corresponding Author) is with the Department of Computer Science, State University of New York at Binghamton, Binghamton, NY 13902. E-mail: {js1002.oh@samsung.com, kang@binghamton.edu}*

takes a control action after a performance problem, such as an excessive data service delay, is detected. In an extreme case, a feedback controller may fail to support the desired performance, if the workload significantly deviates from the operating range used for designing and tuning the controller [11], [12]. As a result, the robustness of real-time data services could be largely impaired.

In this paper, we aim to substantially improve the robustness of real-time data services by supporting not only reactive but also proactive techniques for admission control to avoid large data service delays exceeding the desired delay threshold. A list of our key contributions follows:

- We develop a new approach to adapting the database load bound for admission control in a predictive fashion, making no statistical or queuing-theoretic assumptions about workloads. In this way, we attempt to avoid overload conditions before they occur.
- We seamlessly integrate the predictive method with a feedback-based admission control scheme developed in our previous work [1].
- In addition, we extend Chronos [15]—our QoS-aware database system built atop an open source database [16]—by implementing the integrated predictive-reactive method. The new system is called Chronos-2. We thoroughly evaluate the performance of the integrated predictive-reactive method and three baselines representing the state of the art in Chronos-2. Notably, our work is different from most existing RTDB work not implemented and evaluated in a real database system [14], [17].

In Chronos [15], the feedback controller dynamically adapts the queue length, if necessary, to support the desired data service delay, e.g., $2s$. Generally, a delay bound of a few seconds is realistic. For example, E*TRADE [18] and Ameritrade [19] provide a 2s and 5s execution guarantee, respectively[1]. In these systems, a user is not charged commission for a trade that takes longer than the delay bound. Also, queries and transactions, e.g., stock quotes and sale/purchase transactions, in these database systems are relatively simple and predefined for timeliness [2], [3]. However, although the functionalities of transactions are often predefined in RTDBs, the amount of data to process may change from data service request to request unless every user accesses the same amount of data. Thus, it is more effective to manipulate the amount of data to process, called the database backlog, rather than managing the queue length to closely support the desired data service delay threshold [1]. Based on this observation, we design a database-specific feedback control scheme to adapt the data service backlog bound in the closed-loop, if necessary, to support the desired data service delay threshold even in the presence of dynamic workloads.

Furthermore, we design a new mathematical model to adapt the database backlog bound in a proactive manner, if necessary, to support the desired delay threshold. Our feedforward regulator[2] predicts the impact of transient load changes on the data service delay. Specifically, we extend the geometric model for web server delay prediction [21] that only considers the queue length and sequential processing of web service requests. Our extended model predicts the data service delay based on the database backlog rather than the queue length, while considering concurrent processing of data service requests.

In this paper, the predictive and reactive approaches are integrated as a unified framework for admission control. In the integrated predictive-reactive scheme, an incoming data service request is admitted to the database system, if the estimated total backlog does not exceed the backlog bound after admitting the request. Otherwise, the request is rejected and a system busy message is sent to the client. By adjusting the backlog bound in a proactive manner, the robustness of the feedback control scheme is enhanced due to the reduced burstiness of workloads. Also, the backlog bound is adapted by the feedback control signal computed at the next sampling interval to compensate potential prediction errors due to dynamic workloads. In sum, we integrate the predictive and reactive methods in a synergistic manner to systematically enhance the data service quality in terms of the average/transient data service delay.

To evaluate the effectiveness of the proposed approach, we conduct extensive experiments. Although our predictive and reactive methods for timely data services is not limited to a specific soft RTDB application, our RTDB benchmark models stock trading as a case study. Our benchmark has similar database schema and transactions to TPC-W [22]. Especially, it extends TPC-W by adding periodic temporal data updates needed to meet the data freshness requirements in RTDBs [2], [3]. For consistent performance comparisons, all the tested approaches are implemented in Berkeley DB [16] that is a popular open source database. Our integrated predictive-reactive scheme significantly outperforms the open-loop Berkeley DB that accepts all incoming requests with no feedback or feedforward control, a feedback-only method [1], and a novel approach for timely data services based on statistical feedforward + feedback control [23], which represent the current state of the art.

In all the experiments, our integrated predictive-

1. Automated high frequency trading systems that support milliseconds of delays exist [20]. In this paper, we focus on processing interactive data service requests, similar to [18], [19]. Our approach could be adapted to support a shorter delay bound by either admitting fewer requests or using more computational resources. A thorough investigation of these system design and tradeoff issues is reserved for future work.

2. In this paper, our delay prediction scheme is also called a feedforward regulator to emphasize that it proactively adjusts the database workload to support the desired data service delay in concert with a feedback control scheme.

reactive scheme controls the average data service delay to be below the desired delay threshold. It shows no transient delay overshoots, i.e., transient delays larger than the specified bound, for more than 99% of the time during the experiments. At the same time, the magnitude of the overshoots is negligibly small. Moreover, our approach increases the timely throughput, i.e., the total amount of data processed by timely transactions that finish within the desired delay bound, by up to 92% compared to the feedback-only baseline. Note that our approach is lightweight. We have measured the CPU consumption through the `/proc` interface in Linux. Our integrated predictive-reactive approach for real-time data services consumes less than 2% CPU utilization in total, while using less than 5KB of memory mainly to store the meta data needed to estimate database backlog.

The rest of this paper is organized as follows. The architecture of Chronos-2, admission control scheme, database backlog estimation method, and real-time data service requirements are described in Section 2. A description of our feedforward regulator is given in Section 3. Our feedback control scheme is discussed in Section 4. The performance of the predictive-reactive scheme and baselines is evaluated in Section 5. A discussion of related work is given in Section 6. Finally, Section 7 concludes the paper and discusses future work.

2 Architecture for Timely Data Services

In this section, an overview of Chronos-2 architecture and predictive-adaptive admission control scheme for timely data services is given. Also, the backlog estimation scheme and an example service level agreement (SLA) used for performance evaluation in this paper are described.

2.1 Chronos-2 Architecture

Figure 1 shows the architecture of Chronos-2. It consists of the database backlog estimator, feedforward regulator, feedback controller, admission controller, performance monitor, and a database server. The backlog estimator estimates the total amount of data the database has to process to handle the data service requests, i.e., queries and transactions, currently in the system. The feedforward and feedback controllers adapt the backlog bound, if necessary, to support the desired delay bound in a predictive and reactive manner, respectively. The admission controller admits or rejects an incoming data service request, if the sum of the estimated amount of data to be accessed by the request and the current database backlog does not exceed the backlog bound computed by the feedforward and feedback (F^{3}B) controller.

The database server processes data service requests such as stock quote queries and online trade transactions. Also, it periodically updates stock prices received from the real-time data generator to support the freshness of stock prices. In Chronos-2, dedicated threads are reserved to update stock prices. These threads for periodic updates are always executed to support the data freshness essential for real-time data services [3]. Thus, admission control is only applied to user requests, if necessary, to avoid overload.

Specifically, Chronos-2 periodically updates 3000 stock prices. A fixed update period selected in a range [$0.2s$, $5s$] is associated with each stock price. The schema of the stock price data follows the schema of Yahoo! Finance data [24]. Online stock price tickers are available for free of charge; however, the update frequencies of online tickers such as [24] is generally lower than the update frequencies used in this paper. Also, online tickers usually provide price updates for only a small number of stock items. To support relatively high frequency updates for thousands of data items, we use our own data generator. Note that actual stock prices are not important in this paper, because we are not building a commercial stock trading system but a RTDB testbed.

The database server schedules accepted requests in a FIFO (First-In-First-Out) manner, while applying 2PL (two phase locking) [25] for concurrency control. Our approach can support timely data services without requiring special scheduling support from the underlying real-time operating system, which is not usually employed by e-commerce applications. Since most databases support FIFO and 2PL, our QoS-aware approach, i.e., F^{3}B, is relatively easy to deploy.

2.2 Admission Control via F^{3}B

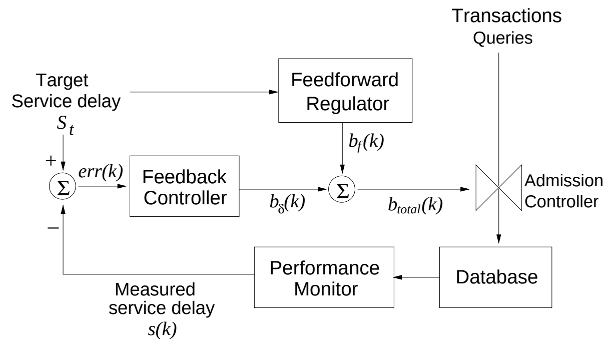

Fig. 2. Feedforward and Feedback Control of the Data Service Delay

For F^{3}B control, we let the $k^{th}(\geq 1)$ sampling period be the time interval $[(k-1)SP, kSP)$. The k^{th} sampling instant is equal to kSP. Let $n(k)$ be the number of the data service requests, i.e., transactions and queries, processed in the k^{th} sampling period. Our F^{3}B admission controller is depicted in Figure 2. Also, a description of its overall behavior follows:

- At the k^{th} sampling instant, the feedforward regulator in Figure 2 predicts the database backlog bound $b_f(k)$ to control the data service delay to be smaller than or equal to the desired delay bound S_t in the $(k+1)^{th}$ sampling period as shown in Figure 3.

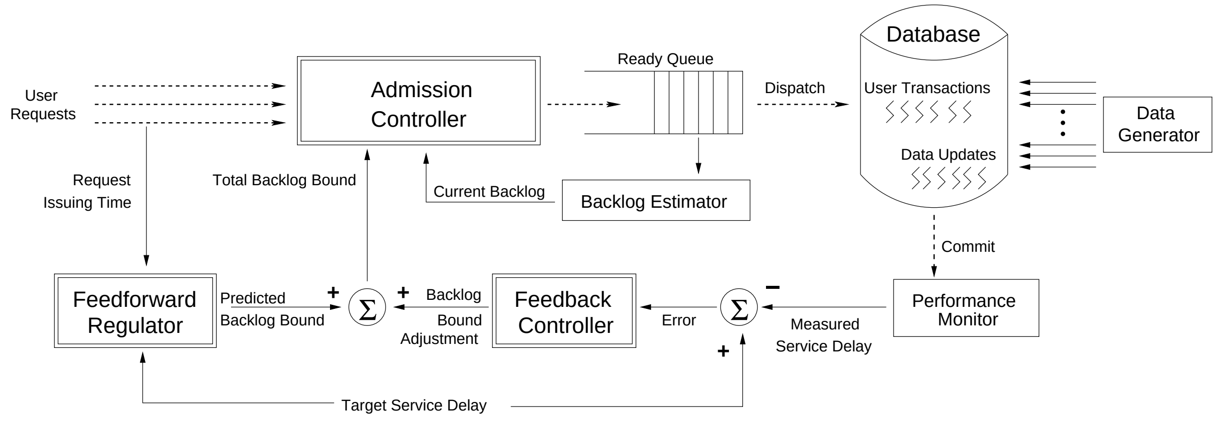

Fig. 1. Chronos-2 QoS-Aware Database Architecture

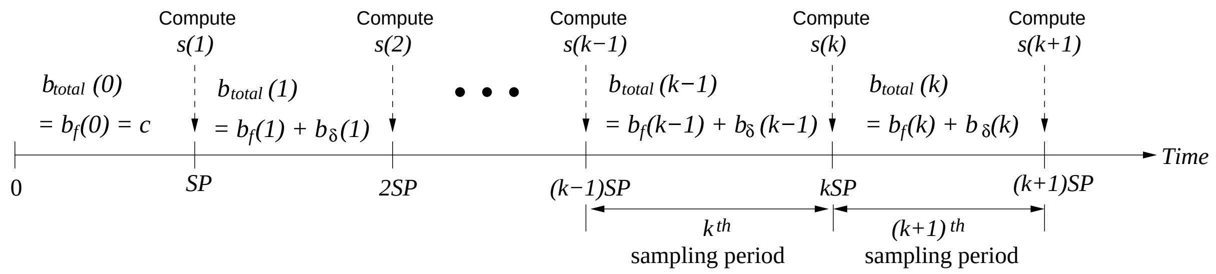

Fig. 3. Computation of the Total Backlog Bound via F^{3}B Control at Each Sampling Instant.

Note that every parameter needed to compute $b_f(k)$ is known at the k^{th} sampling instant. Thus, $b_f(k)$ to be used in the $(k+1)^{th}$ sampling period can be computed in a predictive manner at the k^{th} sampling instant. (A detailed discussion is given in Section 3.) Initially, $b_f(0)$ is set to a small positive constant c.

- At the k^{th} sampling instant, the performance monitor in Figure 2 computes the average service delay $s(k) = \sum_{i=1}^{n(k)} s_i/n(k)$ where s_i is the delay for processing the i^{th} service request finished in the k^{th} sampling period, i.e., the time interval $[(k-1)SP$, $kSP)$ as shown in Figure 3. Specifically, the delay for serving the i^{th} request is: $s_i = c_i + q_i + e_i$ where c_i, q_i, and e_i represent the TCP connection delay, queuing delay, and processing delay for data accesses inside the database by the i^{th} request. Based on the error $err(k) = S_t - s(k)$, the feedback controller (discussed in Section 4) computes the required backlog bound adjustment $b_\delta(k)$. Because no feedback is available yet when the system starts running, we set $b_\delta(0) = 0$. Thus, $b_{total}(0) = b_f(0) = c$.
- Given $b_f(k)$ and $b_\delta(k)$, the admission controller updates the database backlog bound to be used during the $(k+1)^{th}$ sampling period as depicted in Figure 3:

$$b_{total}(k) = b_f(k) + b_\delta(k). \qquad (1)$$

To manage the backlog bound, we also apply the anti-windup technique [26]. If $b_{total}(k) < 0$ we set $b_{total}(k) = 0$. To avoid excessive backlog, we also make $b_{total}(k)$ = *max-data* if $b_{total}(k) >$ *max-data* where *max-data* = *max-queue-size* · *max-transaction-size*, which is the estimated maximum transaction size defined in terms of the amount of data for one transaction to process.
- In the $(k+1)^{th}$ sampling period, the database accepts an incoming data service request T_{new} as long as the estimated backlog in the ready queue in Figure 1 does not exceed $b_{total}(k)$ in Eq 1 after accepting the new request. Eq 2 describes this admission control rule:

$$\text{Admit } T_{new} \text{ if } d_{new} + \sum_{i=1}^{Q_t} d_i \le b_{total}(k) \qquad (2)$$

where d_{new} and d_i are the amount of data to be processed by T_{new} and the i^{th} service request in the ready queue, respectively. Q_t is the total number of requests in the ready queue at time t where $kSP \le t < (k+1)SP$.

For our F^{3}B admission control scheme, we set the sampling period $SP = 1s$ during which several hundreds to thousands of transactions arrive at the database in our Chronos-2 testbed. One may argue that more frequent feedback control using a shorter sampling period could be more effective than F^3B is. However, the measured data service delay may widely vary from sampling period to period, if the feedback controller uses too short a sampling period, in which few service requests can finish. As a result, the performance of the feedback control system may largely fluctuate. Also, frequent sampling increases the overhead. In fact, selecting an optimal sampling period is a difficult problem [11]. Especially, it is harder in a computational system, such as a database system, whose behavior is highly dynamic and stochastic unlike physical systems, such as the cruise control system in automobiles, which can be modeled following physics laws [12]. A thorough investigation of a more advanced approach for selecting a cost-effective sampling period for QoS-aware data services is reserved for future work. All the tested approaches, including ours, applying feedback control theory use the same sampling period for fair performance comparisons in Section 5.

2.3 Backlog Estimation

Our case study considers common types of transactions for online trades: view-stock, view-portfolio, purchase, and sale for seven tables [1]. The database schema and transactions are similar to TPC-W [22], which models a small number of tables and well-defined queries and transactions such as catalog browsing, purchase, and sale in an emulated online bookstore. Further, we add periodic stock price updates for data freshness management as discussed before. We claim that our benchmark is more appropriate than DSMS benchmarks such as TPC-DS [27], since the focus of this paper is to support the desired delay for processing one-time, interactive transactions in the context of RTDBs rather than processing persistent queries in DSMS.

For backlog estimation, we leverage our stock trade database schema and semantics of transactions/queries, which largely affect the data access pattern and amount of data to process. Especially, we focus on a subset of the tables directly related to the database backlog estimation. The information about the schema and transaction types in a database application is usually available. Further, predefined (or canned) transactions can be used to support the timeliness of real-time data service requests to support, for example, target tracking or traffic monitoring [2], [3]. Thus, our approach is applicable to workload estimation in RTDBs. Specifically, for a transaction (or query) T_i from a client, we estimate the amount of data to access as follows.

- **View-stock**: This query is about a set of companies' information and their associated stock prices. To process this request, the database system needs to access STOCKS and QUOTES tables that hold <stock symbol, full company name, company ID> and <company ID, current stock price> for each company. After parsing the query, the system finds the number of companies n_c specified in the query. It then calculates the amount of data to access for T_i: $d_i = n_c \cdot \{r(\text{STOCKS}) + r(\text{QUOTES})\}$ where $r(tbl)$ is the average size of a row (i.e., the average number of bytes in a row) in table tbl.
- **View-portfolio**: In this query, a client wants to see certain stock prices in its portfolio. For each stock item in the portfolio, the system needs to look up the PORTFOLIOS table that holds <client ID, company ID, purchase price, shares> to find the company IDs used to look up the QUOTES table. Thus, the estimated amount of data to access for this query is: $d_i = |portfolio(id)| \cdot \{r(\text{PORTFOLIOS}) + r(\text{QUOTES})\}$ where $|portfolio(id)|$ is the number of stock items in the portfolio owned by the client whose ID is id.
- **Purchase**: When a client places a purchase order for a stock item, the system first gets the current stock price from the QUOTES table. If the purchased stock was not in the portfolio before the purchase, the stock item and its purchase price are added to the portfolio. If it is already in the portfolio, the system updates the corresponding shares. Hence, the estimated amount of data accesses for a PURCHASE transaction is: $d_i = r(\text{QUOTES}) + (|portfolio(id)| + 1) \cdot r(\text{PORTFOLIOS})$.
- **Sale**: A client can sell a subset of its stocks in the portfolio via a sale transaction. In our current implementation, a client simply specifies the number of stocks $n_{i,sell}$ that it wants to sell where $n_{i,sell} \leq |portfolio(id)|$. To process a sale transaction, the system scans the PORTFOLIOS table to find the stock items belonging to this client's portfolio. Using the stock IDs found in the PORTFOLIOS table, the system searches the QUOTES table to find the corresponding stock prices. After that, it updates the client's portfolio in the PORTFOLIOS table to indicate the sale. Thus, the estimated amount of data to process for this transaction is: $d_i = |portfolio(id)| \cdot r(\text{PORTFOLIOS}) + n_{sell} \cdot r(\text{QUOTES}) + n_{sell} \cdot r(\text{PORTFOLIOS})$.

For database backlog estimation, the transaction size is measured in terms of the number of data accesses, e.g., the number of the stock prices to read or the number of portfolio updates, needed to process the transaction. This approach is similar to techniques for evaluating database operators [25]. We take this approach, because our stock trading testbed mainly handles textual data whose size does not significantly vary from row to row (or table to table). Thus, in this paper, the row size is considered uniform. For different types of data such as images, backlog estimation needs to consider the size of individual data items in addition to the number of data to access.

Two kinds of meta data are used for estimation: system statistics and materialized portfolio information. System statistics are used to obtain general information of every table such as the number of the rows, row sizes, and number of distinct keys in the table. We periodically update the statistics at every sampling period. For more accurate backlog estimation, we keep track of the number of the stock items in each user's portfolio via view materialization. In this paper, portfolio information is materialized for backlog estimation; that is, we precount the number of stock items in each user's portfolio and incrementally update the number whenever it is changed. By doing it, we can maintain the consistency between the meta data and the underlying table in the database. When a client wants to look up its portfolio, our backlog estimator can immediately find how many stocks need to be quoted to process this request using the meta data.

2.4 Example Service Level Agreement

TABLE 1
Service Level Agreement

Notation	Description	Desired Performance
S_t	Service Delay	$S_t \leq 2s$
S_v	Delay Overshoot	$S_v \leq 2.5s$
T_v	Settling Time	$T_v \leq 10s$

An SLA can be specified by an administrator of a real-time data service application, such as e-commerce, and agreed with clients. The SLA considered in this paper is given in Table 1. In this SLA, the average delay needs to be shorter than or equal to $S_t = 2s$, because most clients are likely to leave if the service delay becomes longer than a few seconds [4]. We also specify transient performance requirements, because a long-term average delay may not fully represent the quality of data services. The transient service delay is the average of the service delays measured in one sampling period. An overshoot S_v, if any, is a transient service delay longer than S_t. It is desired that $S_v \leq 2.5s$ and it becomes smaller than or equal to S_t within the settling time $T_v = 10s$. In this paper, our feedforward scheme adjusts the backlog bound in a proactive manner, if necessary, to closely support the desired delay threshold S_t. Also, formal control-theoretic techniques [11], [12] are applied to control the desired transient performance characteristics represented by S_v and T_v in Table 1 in addition to supporting S_t. Due to the systematic design, F^{3}B closely supports the SLA given in Table 1 with few overshoots, while significantly improving the timely throughput compared to the tested baselines in our performance evaluation presented in Section 5.

3 Predictive Database Backlog Management

The objective of the feedforward regulator is to avoid potential overload and large delays violating the SLA in a proactive fashion. To model the relationship between incoming data service requests and resulting data service delays, we extend a geometric model for web server delay prediction [21]. In [21], the authors derive a web server delay prediction model by using a diagram that shows the request arrivals at the system, sequential processing of the web service requests, and departures along the time line. Based on the queue length and arrival times of requests, they estimate the future delay for processing the queued requests. The web server then dynamically adapts the processing speed of the CPU via dynamic voltage scaling to save energy, while supporting the target delay. The advantage of the geometric model [21] is its independence of a specific queuing or statistical model. However, it only considers the queue length and sequential request processing. In this section, we develop a new model for database delay prediction that analyzes the database backlog rather than the queue length, while taking concurrent processing of data service requests into account.

Figure 4 illustrates an example of processing data service requests in a database. In Figure 4, the connection delay c_i, queuing delay q_i, and processing delay e_i are depicted by an unfilled, dashed, and filled rectangle, respectively. Table 2 summarizes the notations used to describe our delay prediction model. In Chronos-2, an incoming request is added to the ready queue in Figure 1 as it arrives. To support concurrent processing of requests, Chronos-2 dequeues and schedules the first request in the queue as soon as a thread becomes available.

At time x in Figure 4, three transactions are executed concurrently; therefore, $P_x = 3$. The number of remaining transactions queued waiting to be processed is: $Q_x = 6$. As a result, the total number of requests in the system at time x is 9; that is, $N_x = P_x + Q_x = 9$ in Figure 4. Based on this geometric model, we derive the equation that describes the relationship between the service delay and database backlog as follows. Let $\hat{s}$ be the average service delay of N_x requests; that is, $\hat{s}$ is the average length of N_x horizontal bars in Figure 4. The area occupied by N_x bars, i.e., the polygon BFGD, indicates the total service delay experienced by the N_x requests, which is equal to $\hat{s} \cdot N_x$ where $\hat{s}$ is the average delay to be computed. The area of BFGD can be approximated as the sum of two polygon areas divided by the vertical line $\overline{CG}$: a polygon BFGC forming the area on the left side of $\overline{CG}$ and a polygon CGD occupying the area on the right side of $\overline{CG}$. The area of the polygon BFGC can be obtained by subtracting the area of AEFB from the area of rectangle AEGC. Therefore, the area of BFGD is geometrically expressed by:

$$BFGD \quad \approx \quad AEGC - AEFB + CGD \tag{3}$$

The **area AEGC** in Figure 4 is $\overline{EG} \cdot \overline{CG}$ where the length of $\overline{EG}$ is $x - t_s$ that indicates the difference between the current time and the time at which the database system started to operate. In the rest of the paper, we assume $t_s = 0$ for the clarity of presentation. The length of $\overline{CG}$ is N_x, i.e., the number of data service requests in the system. Thus, the area of AEGC is $x \cdot N_x$.

The **area AEFB** is the area between the y-axis and the starting point of each horizontal bar in Figure 4, similar to a trapezoid. The starting point of the i^{th} bar indicates the time at which the i^{th} request is issued at the client side. Let t_i be the starting point of the i^{th} bar. Given that, the area of AEFB can be computed as $\frac{1}{2}N_x(t_1 + t_{N_x})$.

TABLE 2
Variables Used for Feedforward Regulator

Notation	Description
s_i	Service delay for the i^{th} service request
$\hat{s}$	Average service delay
c_i	Connection delay for the i^{th} service request
q_i	Queuing delay for the i^{th} service request
e_i	Processing delay for the i^{th} service request
d_i	Number of data accesses for the i^{th} service request
t_i	Request issuing time for the i^{th} service request, i.e., the beginning of the connection delay c_i
ρ	Estimated processing delay per data access
b_f	Predicted backlog bound computed by the feedforward regulator
P_x	Number of service requests being processed at time x
Q_x	Number of service requests in the ready queue at time x
N_x	Number of service requests in the system, i.e., $P_x + Q_x$
R_i	The i^{th} service request in the system

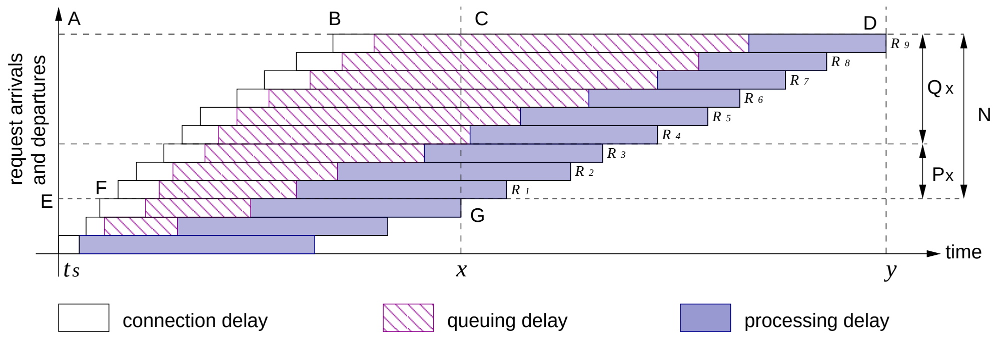

Fig. 4. Delay Prediction Model

Further, the **area CGD** can be approximated as a triangle. Thus, its area can be computed as $\frac{1}{2}(\overline{CG} \cdot \overline{CD})$. The length of the vertical line $\overline{CG}$ is simply the number of requests in the system, i.e., N_x. The length of the horizontal line $\overline{CD}$ is the time interval between x and y, i.e., $y - x$ in Figure 4. Thus, the area of BFGD, $\hat{s} \cdot N_x$, is:

$$\hat{s} \cdot N_x \approx x \cdot N_x - \frac{1}{2} \cdot N_x \cdot (t_1 + t_{N_x}) + \frac{1}{2} \cdot N_x \cdot (y - x) \quad (4)$$

By dividing Eq 4 by N_x, we get:

$$\hat{s} \approx x - \frac{1}{2}(t_1 + t_{N_x}) + \frac{1}{2}(y - x) \quad (5)$$

The term $(y - x)$ in Eq 5 indicates the total time the database needs to spend to serve N_x requests as shown in Figure 4. To predict the term $(y - x)$ precisely, we should have *a priori* knowledge of the blocking delay, if any, caused due to potential data/resource contention among transactions concurrently executed. However, precisely predicting potentially time-varying blocking delays in a database system is very hard, if at all possible. Instead, we measure the average delay for a single data access, ρ, from a separate experiment conducted offline. Given ρ and the estimated number of data to be accessed by each transaction analyzed using the backlog estimation technique described in Section 2.3, we can estimate how much time will take to process a transaction. In this way, we estimate $(y-x) = \rho \cdot \sum_{i=1}^{N_x} d_i$ where N_x is the number of transactions in the database system and d_i is the estimated number of data to be accessed by the i^{th} data service request. Therefore, Eq 5 becomes the following:

$$\hat{s} \approx x - \frac{1}{2}(t_1 + t_{N_x}) + \frac{1}{2}\rho \sum_{i=1}^{N_x} d_i \quad (6)$$

This approach is optimistic in that the actual data access delay in Figure 4 may increase due to data/resource contention among concurrent transactions. However, we intentionally take this optimistic approach to avoid potential underutilization of the database system due to pessimistic predictions. As many databases support 2PL or variants, the optimistic nature of our approach could compensate for potential pessimism about data conflicts introduced by lock-based concurrency control. Further, we apply a feedback-based reactive approach to adjust the backlog bound, if necessary, to support the desired delay threshold.

By solving Eq 6 for $\sum_{i=1}^{N_x} d_i$, we get:

$$\sum_{i=1}^{N_x} d_i \approx \frac{2}{\rho}\{\hat{s} - x + \frac{1}{2}(t_1 + t_{N_x})\} \quad (7)$$

To estimate the backlog bound, i.e., the amount of data to be processed for the requests in the ready queue, we subtract the number of data accesses that the database is currently processing, i.e., $\sum_{i=1}^{P_x} d_i$ in Figure 4, from Eq 7.

Finally, by replacing $\hat{s}$ in Eq 7 with the desired average service delay bound S_t, we compute the estimated backlog bound at the k^{th} sampling instant, i.e., time x in Figure 4:

$$\begin{aligned} b_f(k) &\approx \sum_{i=1}^{N_x} d_i - \sum_{i=1}^{P_x} d_i \\ &\approx \frac{2}{\rho}\{S_t - x + \frac{1}{2}(t_1 + t_{N_x})\} - \sum_{i=1}^{P_x} d_i \end{aligned} \quad (8)$$

that will be used to support the delay threshold S_t via admission control in the $(k+1)^{th}$ sampling period. To summarize, in Eq 8, ρ is derived offline, S_t is given by the SLA, x is the current time, and t_1 and t_{N_x} are the issuing times of the request 1 and request N_x. Further, $\sum_{i=1}^{P_x} d_i$ is derived via the backlog estimation scheme. As all the variables in Eq 8 are known at the k^{th} sampling point, the feedforward controller can derive $b_f(k)$ to be used in the $(k+1)^{th}$ sampling period without having to wait for a feedback to be computed at the $(k+1)^{th}$ sampling point.

4 Feedback Control of Data Service Delays

In addition to feedforward control of the backlog bound, we apply feedback control techniques [11], [12] to support the desired delay bound even in the presence of dynamic workloads. We apply formal control theory to design a feedback controller for timely data services in a stepwise manner:

1) Model the relation between the backlog and data service delay.
2) Design and tune the feedback controller.
3) Evaluate the performance. If the desired performance is not supported, redo the previous steps.

A detailed description of system modeling and controller design follows.

4.1 System Modeling

It is essential to model the controlled system, such as the database system, to design a feedback control system that can support the desired performance even when dynamic workloads are given. Our control objective is to support the desired data service delay by regulating the database backlog bound.

Intuitively, the data service delay increases as the backlog increases and vice versa. To capture the relation between the backlog and data service delay, we employ an ARX (Auto Regressive eXternal) model [11], [12]. Let $s(k)$ denote the average delay of the transactions and queries that finished during the k^{th} sampling period and $b(k)$ be the average database backlog measured in the same sampling period. We build an $n^{th}(n \geq 1)$ order difference equation that models $s(k)$ using the n previous service delays and database backlogs:

$$s(k) = \sum_{i=1}^{n}\{\alpha_i s(k-i) + \beta_i b(k-i)\} \quad (9)$$

where $s(k-i)$ is the average service delay of the data service requests finished in the $(k-i)^{th}$ sampling period and $b(k-i)$ is the average database backlog in the period. Using this difference equation, we model database dynamics considering individual data service requests, potentially having different amounts of data to process. Thus, our control model is fine-grained.

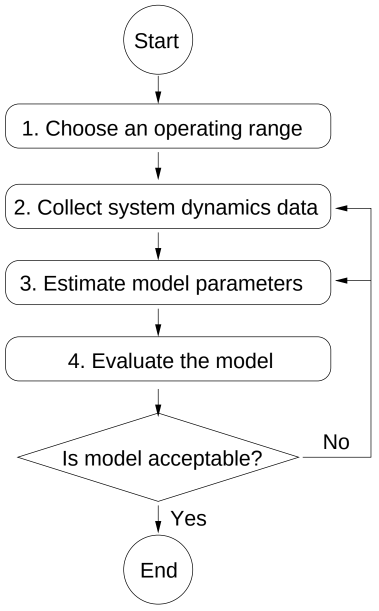
Start
1. Choose an operating range
2. Collect system dynamics data
3. Estimate model parameters
4. Evaluate the model
Is model acceptable?
No
Yes
End

Fig. 5. Steps in System Identification (SYSID)

In Eq 9, $n(\geq 1)$ is the system order [12]. Generally, a higher order model is more accurate but more complex. To derive the unknown model coefficients α_i's and β_i's in Eq 9 and determine the system order n in Eq 9, we apply the system identification (SYSID) method [11], [12], which consists of the four steps shown in Figure 5 and described in the following.

Step 1. Choose the model scope (operating range): For SYSID, we have observed how the database response time changes for the decreasing inter-request time, which is the time interval between two consecutive requests, using the stock trade benchmark. 1200 client threads concurrently send data service requests to the database that processes user data service requests, while periodically updating 3000 stock prices as discussed before. Each client sends a service request and waits for the response from the database server. After receiving the transaction or query processing result, it waits for an inter-request time uniformly selected in a range before sending the next data service request.

We have found that the service delay measured in our database testbed shows a near linear pattern around $S_t = 2s$ when the inter-request time ranges between $0.5s$

and $3.5s$. For the inter-request time longer than $3.5s$, the response time is under $0.01s$. As the relation between inter-request time and response time is near linear, we can apply the well-established linear control theory [11]. For unbiased parameter estimation via SYSID, the inter-request time is randomly selected within the operating range [$0.5s$, $3.5s$]. Thus, the 1200 client threads submit approximately $342 - 2,400$ requests per second to the system.

Step 2. Collect system dynamics data: For SYSID, it is important to collect and analyze a large amount of data for system modeling. To collect a large number of performance data, we run an experiment for 60 minutes and measure backlog and service delay at every second. By doing this, we moderate the stochastic nature of a database. Specifically, for accurate SYSID of statistical database dynamics, we use the square root values of the performance data, similar to [28].

Step 3. Estimate model parameters: We apply the recursive least square method [11], [12] to estimate model parameters in Eq 9 by minimizing the sum of the squared prediction errors. Specifically, the prediction error in our model $e(k) = s(k) - s'(k)$, where $s(k)$ is the actual response time measured at the k^{th} sampling instant and $s'(k)$ is the expected response time computed by Eq 9 using the estimated model parameters available at the $(k-1)^{th}$ sampling instant. The recursive least square method iteratively estimates the model parameters at each sampling instant to minimize the sum of the squared estimation errors $\sum_{k=1}^{m} e(k)^2$ where m is the number of the sample instances.

Step 4. Evaluate the accuracy of the model: To analyze the accuracy of SYSID, we use the R^2 metric [12]. In this paper, $R^2 = 1-$ variance(service delay prediction error)/variance(actual service delay). A control model is acceptable if its $R^2 \geq 0.8$ [12]. We have derived the $1^{st}, 2^{nd}, 3^{rd}$, and 4^{th} order models. To design a feedback controller, we choose the 4^{th} order model that shows the highest $R^2 = 0.99$ among the tested models. Specifically, the derived 4^{th} order model captures the relation between the database backlog and delay as follows:

$$\begin{aligned} s(k) &= 0.1015s(k-1) + 0.2767s(k-2) \\ &+ 0.4595s(k-3) + 0.0354s(k-4) \\ &+ 0.0135b(k-1) + 0.0003b(k-2) \\ &- 0.0046b(k-3) - 0.0074b(k-4). \end{aligned} \tag{10}$$

4.2 Feedback Controller Design

The relation between the database backlog and the data service delay may change over time; therefore, the backlog bound may need to be updated to support the desired data service delay via feedback control. As the first step to designing the feedback controller, we derive the transfer function of the open-loop database by taking the z-transform [12] of Eq 10. z-transform converts a discrete time domain signal to the frequency domain value that can be manipulated algebraically. In this paper, the service delays $s(k), ..., s(k-3)$ and backlogs $b(k), ..., b(k-3)$ in Eq 10 are converted to $S(z), ..., z^{-3}S(z)$ and $B(z), ..., z^{-3}B(z)$ in the frequency domain where z is the delay parameter. After some algebraic manipulation, we derive the following transfer function that shows the relation between the data service delay and backlog:

$$\begin{aligned} \tau(z) &= \frac{S(z)}{B(z)} \\ &= \frac{0.0135z^3 + 0.0003z^2 - 0.0046z - 0.0074}{z^4 - 0.1015z^3 - 0.2767z^2 - 0.4595z - 0.0354} \end{aligned} \tag{11}$$

Given the system model expressed in Eq 11, we can use a standard feedback controller, such as the proportional-integral-derivative (PID) controller [11], [12], to implement the backlog bound controller. In this paper, we use a PI controller. A P controller cannot remove the steady state error [11], [12]. An integral controller can be combined with a P controller to eliminate the non-zero steady-state error [11], [12]. A derivative controller could decrease the settling time by adjusting the control input based on changes in control errors. However, a derivative controller is sensitive to noise [11], [12]. As database workloads are dynamic, we do not use a D controller in this paper.

At the k^{th} sampling instant, the PI controller computes the control signal $b_\delta(k)$, i.e., the database backlog adjustment needed to support S_t using the following PI control law:

$$b_\delta(k) = b_\delta(k-1) + K_P[(K_I+1)err(k) - err(k-1)] \tag{12}$$

where the error $err(k) = S_t - s(k)$ at the k^{th} sampling instant as shown in Figure 2. The z-transform of Eq 12 is:

$$\psi(z) = \frac{\Delta B(z)}{E(z)} = \frac{K_P(K_I+1)[z - 1/(K_I+1)]}{z-1} \tag{13}$$

where $\Delta B(z)$ and $E(z)$ are the z-transform of $b_\delta(k)$ and $err(k)$, respectively.

Given the open loop transfer function in Eq 11 and the transfer function of the PI controller in Eq 13, the transfer function of the feedback control system (excluding the feedforward controller) in Figure 2 is: $\eta(z) = \frac{\tau(z)\psi(z)}{1+\tau(z)\psi(z)}$ [11], [12]. Using the closed loop transfer function, we tune the controller via the Root Locus method in MATLAB [11], [12], in which one can graphically tune the controller to support the desired average and transient service delay specified in Table 1 and the stability of the closed-loop system. It is required to locate the closed loop poles, i.e., the roots of the denominator of $\eta(z)$, inside the unit circle to support the stability [11], [12]. Also, we locate the poles to support the desired average/transient performance specified in Table 1. The selected PI controller coefficients are $K_P = 4.93$ and $K_I = 5.36$. $b_\delta(k)$ computed in the closed-loop system is used to compute the backlog bound $b_{total}(k)$ in Eq 1.

Due to our system modeling and controller tuning, the feedback-based admission controller can support the desired delay bound. However, a feedback controller may fail to support the desired performance, if the workload, e.g., the arrival rate of data service requests, largely deviates from the operating range used for SYSID [11], [12]. To address the problem, in Chronos-2, our feedforward controller helps the closed-loop system to stay in the operating range by adjusting the backlog bound in a proactive manner. Thus, the feedforward and feedback approaches implemented in Chronos-2 benefit each other as discussed before.

TABLE 4
Tested Workloads

	Number of Clients	Number of requests per second $0s \sim 300s \rightarrow 301s \sim 900s$
W1	1200	[300, 342] $\rightarrow$ [2,400, 12,000] ($5L_H$)
W2	1500	[375, 428] $\rightarrow$ [3,000, 15,000] ($6.25L_H$)
W3	1800	[450, 514] $\rightarrow$ [3,600, 18,000] ($7.5L_H$)
W4	2100	[525, 600] $\rightarrow$ [4,200, 21,000] ($8.75L_H$)
W5	2400	[600, 685] $\rightarrow$ [4,800, 24,000] ($10L_H$)

5 Performance Evaluation

In this section, experimental settings are described. After that, the performance of F^{3}B is compared to the performance of three baselines that represent the current state of art. The objective of this section is to observe how closely the tested approaches can support the desired performance specified in Table 1.

5.1 Experimental Settings

TABLE 3
Tested Approaches

Approach	Feedback	Feedforward
Open	$\times$	$\times$
FB	$\circ$	$\times$
CLT+FB	$\circ$	$\circ$ (statistical)
F^{3}B	$\circ$	$\circ$

Baselines. In this paper, we compare the performance of F^{3}B to the performance of an open-loop database system, a feedback-only system, and a statistical feedforward + feedback system described next and summarized in Table 3:

- **Open** is the basic Berkeley DB [16]. It accepts all incoming requests with neither feedback nor feedforward control. For performance comparisons, all the other tested approaches are implemented atop Berkeley DB.
- **FB** is the feedback-based admission control scheme discussed in Section 4. Hence, FB represents feedback-only approaches for timely data services including [1], [13]–[15].
- **CLT+FB** [23] integrates a statistical feedforward approach based on the central limit theorem [29] and feedback control method for predictive and reactive adjustment of the database backlog bound to support timely data services. To the best of our knowledge, [23] is the only approach developed before F^{3}B to integrate predictive and feedback control methods for real-time data services.

Workloads. For performance evaluation using the stock trading benchmark, we use three identical desktop PCs. Each of them has the dual core 1.6 GHz CPU and 1GB memory. Every machine runs the 2.6.26 Linux kernel. The database server, clients, and stock price generator run on each of them and they are connected through a 1 Gbps Ethernet switch using the TCP protocol. For 60% of time, each client issues a query about current stock prices, because a large fraction of requests can be stock quotes in online trades. For the remaining 40% of time, a client uniformly requests a portfolio browsing, purchase, or sale transaction at a time.

To evaluate the robustness and reliability of our approach in the presence of dynamic workloads, we use five different workloads as shown in Table 4. The number of client threads is increased from 1200 to 2400 by 300 to generate workloads W1 $-$ W5. For all the tested workloads, the inter-request time, i.e., time between two consecutive requests, is uniformly distributed in [$3.5s$, $4.0s$] at the beginning of an experiment. At $300s$, the range of the inter-request time is suddenly reduced to [$0.1s$, $0.5s$] to model bursty workload surges. As a result, the request arrival rate abruptly increases by $7 - 40$ times at $300s$. The reduced inter-request time range is maintained until the end of an experiment at $900s$.

Recall that the operating range used for SYSID in Section 4.1 consists of $342 - 2400$ requests/s. If we let $L_H = 2400$, W1 generates up to $5L_H$ requests/s and W5 creates up to $10L_H$ requests/s as summarized in Table 4. As these workloads largely exceed the model scope (i.e., the operating range), a feedback controller alone may or may not be able to support the desired performance [11], [12], i.e., the SLA specified in Table 1. From these experiments, we intend to observe whether or not F^{3}B enhances the robustness of real-time data services compared to the tested baselines even when the workload largely exceeds the operating range.

For each workload, we run one experiment for $900s$. Each performance data is the average of 10 runs using different random seeds. For the average performance data presented in this paper, we derived the 90% confidence intervals plotted as vertical bars in the graphs.

5.2 Performance Results for Workload 1

Figure 6 shows the average service delay measured in the interval [$301s$, $900s$] of all the tested approaches for W1. Before $301s$, the average service delay is less than $1.2s$ for all the tested approaches due to the relatively low workload. Thus, in this paper, we only present the average delay measured between $301s$ and $900s$ for W1 $-$ W5. (The transient delay is shown for the entire $900s$.)

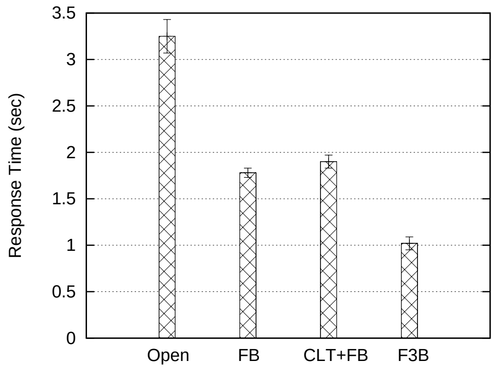

Fig. 6. Average Service Delay for W1

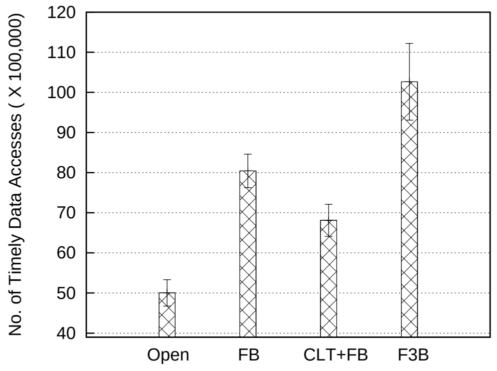

Fig. 7. Timely Throughput for W1

As shown in Figure 6, even for the lightest tested workload, Open cannot support $S_t = 2s$. Further, the transient delay of Open fluctuates between $3s$ and $4s$. This is because Open simply accepts all incoming requests regardless of the current system status. As a result, the system is overloaded. In contrast, the average service delays of the other approaches are not higher than $S_t = 2s$.

By avoiding overload conditions, as shown in Figure 7, FB, CLT+FB, and F^{3}B significantly increase the timely throughput compared to Open. As shown in the figure, Open processes $5,003,833$ timely data in total, which is only 62% and less than 50% of the timely throughput of FB and F^{3}B. As Open fails to support the desired data service delay even for the lightest workload and processes a substantially smaller number of data in a timely manner, we omit its performance results in the remainder of this paper.

Notably, CLT+FB's timely throughput [23] is lower than FB's. This is because it assumes that transactions/queries are independent random variables to apply the central limit theorem [29] for data service delay prediction. Given concurrent transactions and queries, their service delays may not be independent. As our workloads involve more transactions and larger load surges than the workloads of [23] do, the prediction based on the central limit theorem becomes less accurate, resulting in performance degradation. F^{3}B does not make such a statistical assumption about workloads.

5.3 Average Delay and Timely Throughput for Increasing Workloads

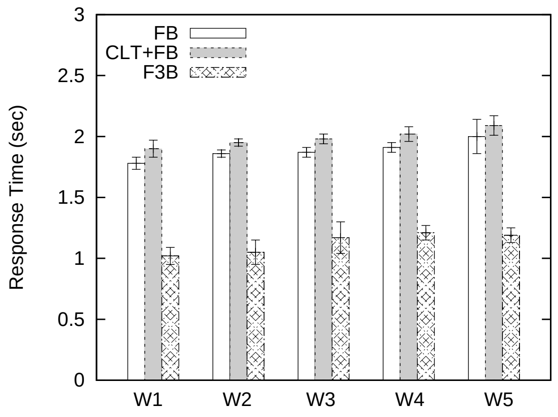

Fig. 8. Average Service Delay

In Figure 8, the average service delay generally increases as the workload increases. As shown in the figure, FB's average service delay ranges in $[1.7s,\ 2.04s]$. The average delay of CLT+FB ranges in $[1.9s,\ 2.09s]$. F^{3}B reduces the average delay to the range of $[1.02s,\ 1.19s]$ by applying proactive backlog management in addition to feedback control. Although FB and CLT+FB closely support the desired average delay as shown in Figure 8, its transient service delay frequently violates the S_t. Unlike FB and CLT+FB, F^{3}B hardly suffers delay overshoots. (Transient data service delays are discussed in detail in Section 5.4.) For these reasons, as shown in Figure 9, F^{3}B significantly outperforms FB and CLT+FB in terms of the timely throughput too. Specifically, F^{3}B processes $27\% - 92\%$ more data in a timely manner than FB does.

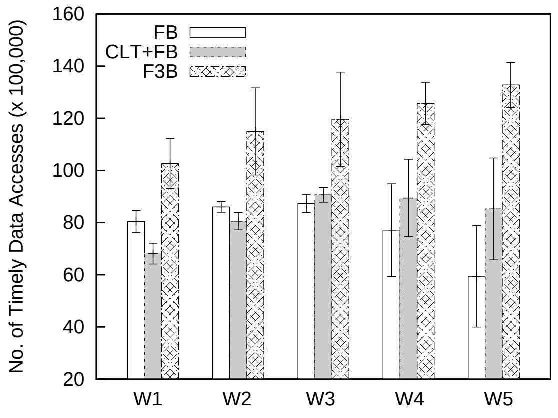

Fig. 9. Timely Throughput

The timely throughput of FB, CLT+FB, and F^{3}B increases as the workload increases up to W3. However, as shown in Figure 9, the throughput of FB decreases sharply as the workload increases beyond W3. As the load increases substantially exceeding the operating range used for SYSID (Section 4), the effectiveness of feedback control decreases. As a result, the system gets overloaded and the delay increases. Accordingly, the timely throughput of FB decreases. In contrast, the number of timely data accesses of CLT+FB and especially F^{3}B generally increases as the workload increases, showing the relative advantage of the predictive-adaptive approaches compared to FB, which is feedback only.

5.4 Transient Delay for Increasing Workloads

TABLE 5
Number of Overshoots

Approaches	Workloads				
	W1	W2	W3	W4	W5
FB	188	243	247	267	387
CLT+FB	410	474	516	508	570
F^{3}B	0	0	1	1	0

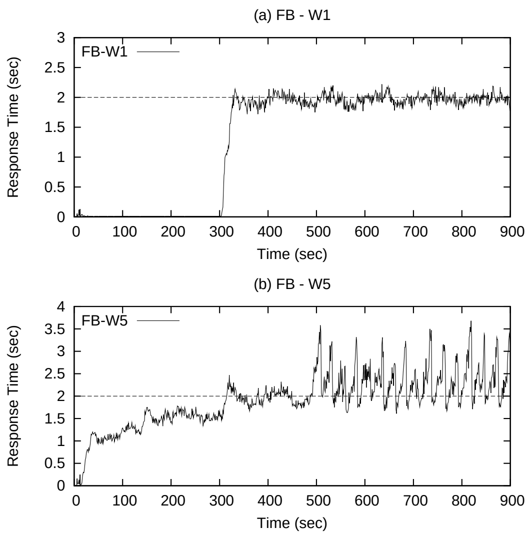

Fig. 10. Transient Service Delay of FB

Figures 10, 11, and 12 plot the transient delay of FB, CLT+FB, and F^{3}B measured at every $1s$ sampling period. Due to space limitations, we only show the transient delays for W1 and W5. Also, Table 5 summarizes the number of overshoots of FB, CLT+FB, and F^{3}B. As shown in Table 5, Figure 10, and Figure 11, FB and CLT+FB suffer many overshoots. However, from the table and Figure 12, we observe that F^{3}B controls the delay to be below $S_t = 2s$ for more than 99.95% of the time. Also, the magnitude of the overshoots are negligibly small as shown in Figure 12.

Interestingly, CLT+FB shows a larger number of overshoots than FB does as summarized in Table 5, but the magnitude of its overshoots is generally smaller than that of FB as shown in Figures 10 and 11. Especially, CLT+FB tends to show better performance than FB does

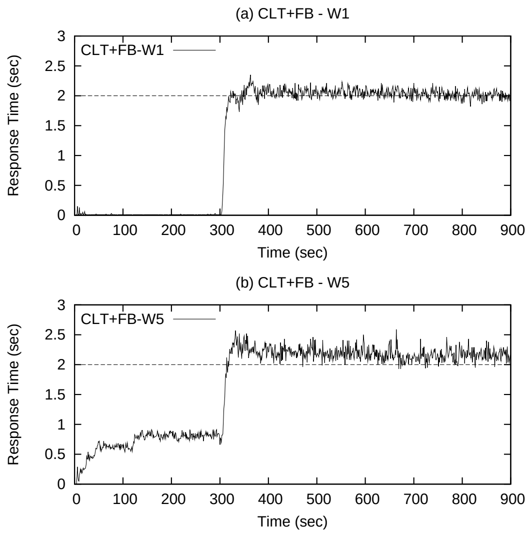

Fig. 11. Transient Service Delay of CLT+FB

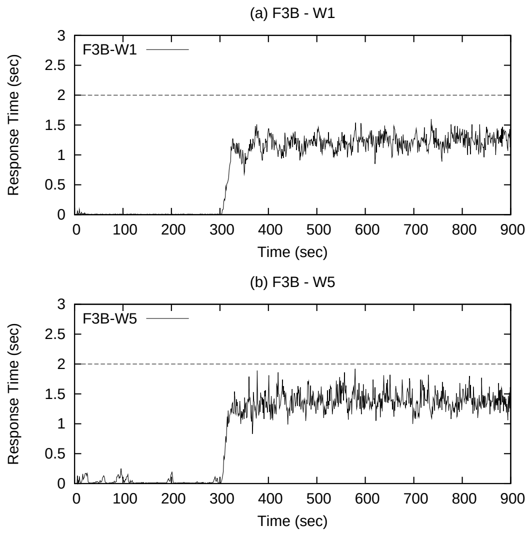

Fig. 12. Transient Service Delay of F^{3}B

as the load increases, since CLT+FB adjusts the backlog bound in a proactive fashion in addition to feedback control of the bound. From Figures 10, 11, and 12, we observe that an overshoot $S_v \leq 2.5s$ (specified in Table 1) for most of the time in CLT+FB and F^{3}B. However, the overshoots in FB considerably exceeds the 2.5s bound as the workload increases. These results show the effectiveness of integrating predictive and reactive methods for timely data services.

Overall, F^{3}B is substantially more robust than FB and CLT+FB due to the synergy created between feedback control and feedforward regulation, which is more accurate than the statistical prediction of CLT+FB [23].

6 Related Work

A common approach for load shedding in DSMS is randomly dropping data tuples under overload according to a certain drop probability, which is normally computed in proportion to the difference between the load and system capacity [9], [30]–[32]. In RTDBs, applying admission control to user queries and transactions is more effective for overload management, since computational resources are reserved to meet the deadlines of periodic temporal data updates [2], [3]. In our previous work [1], we have implemented an RTDB admission control scheme similar to the aforementioned load shedding method in DSMS. The scheme adapts the rate of admitting transactions/queries in proportion to the difference between the current and target delay without applying control theoretic techniques. In the performance evaluation, we compared the performance of this approach to a closed-loop approach for admission control based on formal control theory [11], [12] (similar to FB in Section 5). The latter substantially outperformed the former in terms of the average/transient delay, because it systematically adapts workloads in a closed-loop considering the time-varying load and system status by applying well established control theoretic techniques [1]. In this paper, we further enhance the performance in terms of the delay and timely throughput for real-time data services by integrating a proactive approach with a reactive control-theoretic admission control scheme.

In general, DSMS and RTDB research may benefit from each other. A novel method [10] provides highly accurate estimates of the worst case delay for a given arrival rate of stream data in an event processing system. This delay estimation method and F^{3}B could be integrated to support a desired delay bound in DSMS via delay estimation and systematic load shedding to compensate for potential estimation errors. As relatively little work has been done to support a timeliness guarantee in DSMS, there are many remaining research issues to explore. In addition, other load shedding methods, such as the semantic load shedding algorithm [9], can be applied to favor requests of more important clients in terms of admission control to increase the total utility accrued in an RTDB. A thorough investigation of these important and challenging issues is beyond the scope of this paper and reserved for future work.

Feedback control has been applied to manage the performance of computing systems due to its robustness. It has been employed for real-time scheduling [33], a delay guarantee in a web server [34], differentiated web caching [35], and RPC queue management in Lotus Notes server [36]. However, they do not focus on real-time data service issues such as the timeliness of data services and data freshness. Feedback control has also been applied to support real-time data services [14], [17]; however, their control models are not specific to databases but similar to [33]. Also, these approaches are not implemented in a real database system. Existing approaches for real-time data services, including [1], [15], [37], apply formal control-theoretic techniques [11], [12] to develop feedback-only methods for real-time data services and evaluate them in a real database system. In [23], a feedback + feedforward approach for timely data services is developed. However, it relies on the assumption that the data service delays are independent unlike F^{3}B.

Queuing-theoretic approaches have been applied to predict the web server delay and integrated with feedback control for web server performance management [38], [39]. However, queuing theory requires *a priori* knowledge of the workload model, e.g., the Poisson workload model, which may not be valid for bursty workloads. Schroeder et al. [40] determine an appropriate number of concurrent transactions to execute in a database by applying queuing and control-theoretic techniques. However, a detailed design of the feedback controller is not discussed. Hence, we cannot replicate the approach for comparisons.

7 Conclusions and Future Work

In this paper, we present a new predictive-reactive approach to supporting timely data services even in the presence of dynamic workloads. A summary of our contributions follows.

First, we develop a new method to adapt the database backlog bound in a predictive fashion. Especially, we consider the database backlog rather than the queue length to support the desired delay threshold, while considering concurrent transaction processing.

Second, we present a reactive control-theoretic scheme to adjust the backlog bound, if necessary, to support the desired timeliness of data services. In the integrated admission control framework, the predictive and reactive approaches cooperate to enhance the robustness of real-time data services.

In addition, we have implemented the integrated predictive-reactive scheme in a real database system and undertaken an extensive performance study. Our system significantly reduces the service delay, while enhancing the timely throughput compared to the tested baselines. Overall, our approach is lightweight in terms of the CPU cycle and memory consumption.

Our work is different from most existing work on real-time data management in that it applies both a feedback-based approach and a predictive method to enhance the timeliness of data services. Also, it is implemented and thoroughly evaluated in a real database system unlike most RTDB work, which is not implemented and evaluated in a real database system. In the future, we will continue to explore more effective predictive or reactive approaches as well as workload adaptation techniques for real-time data services.

Acknowledgment

This work was supported, in part, by NSF grant CNS-1117352.

References

[1] K.-D. Kang, J. Oh, and Y. Zhou, "Backlog Estimation and Management for Real-Time Data Services," in *Proceedings of the 20th Euromicro Conference on Real-Time Systems*, 2008, pp. 289–298.

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

[3] K. Ramamritham, S. H. Son, and L. C. Dipippo, "Real-Time Databases and Data Services," in *Real-Time Systems*, vol. 28, 2004, pp. 179–215.

[4] N. Bhatti, A. Bouch, and A. Kuchinsky, "Integrating User-Perceived Quality into Web Server Design," in *Proceedings of the 9th International World Wide Web Conference*, 2000, pp. 1–16.

[5] "Microsoft SQL Server 2008 R2 - StreamInsight," http://www.microsoft.com/sqlserver/2008/en/us/r2-complex-event.aspx.

[6] "Exploratory Stream Processing Systems," http://domino.research.ibm.com/comm/research_projects.nsf /pages/esps.index.html.

[7] "StreamBase," http://www.streambase.com/.

[8] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik, "The Design of the Borealis Stream Processing Engine," in *CIDR*, 2005.

[9] N. Tatbul, U. Cetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker, "Load Shedding in a Data Stream Manager," in *International Conference on Very Large Data Bases*, 2003.

[10] B. Chandramouli, J. Goldstein, R. Barga, M. Riedewald, and I. Santos, "Accurate Latency Estimation in a Distributed Event Processing System," in *International Conference on Data Engineering*, 2011.

[11] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems (3rd edition)*. Addison-Wesley, 1998.

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

[13] M. Amirijoo, N. Chaufette, J. Hansson, S. H. Son, and S. Gunnarsson, "Generalized Performance Management of Multi-Class Real-Time Imprecise Data Services," in *Proceedings of the 26th IEEE International Real-Time Systems Symposium*, 2005, pp. 38–49.

[14] K. D. Kang, S. H. Son, and J. A. Stankovic, "Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 10, pp. 1200–1216, 2004.

[15] K. D. Kang, J. Oh, and S. H. Son, "Chronos: Feedback Control of a Real Database System Performance," in *Proceedings of the 28th IEEE Real-Time Systems Symposium*, 2007, pp. 267–276.

[16] "Oracle Berkeley DB Product Family, High Performance, Embeddable Database Engines," available at http://www.oracle.com/database/berkeley-db/index.html.

[17] M. Amirijoo, J. Hansson, and S. H. Son, "Specification and Management of QoS in Real-Time Databases Supporting Imprecise Computations," *IEEE Transactions on Computers*, vol. 55, no. 3, pp. 304–319, 2006.

[18] "E*TRADE 2-Second Execution Guarantee," https://us.etrade.com/e/t/invest/apptemplate?gxml=exec_guarante

[19] Ameritrade, "Spot and seize potential market opportunities," http://www.tdameritrade.com/powerfultools.html.

[20] N. Y. Times, "Stock Traders Find Speed Pays, in Milliseconds," http://www.nytimes.com/2009/07/24/business/24trading.html.

[21] D. Henriksson, Y. Lu, and T. Abdelzaher, "Improved Prediction for Web Server Delay Control," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, 2004, pp. 61–68.

[22] "Transaction Processing Performance Council," http://www.tpc.org/.

[23] Y. Zhou and K.-D. Kang, "Integrating Proactive and Reactive Approaches for Robust Real-Time Data Services," in *Proceedings of the 30th IEEE Real-Time Systems Symposium*, 2009, pp. 105–114.

[24] "Yahoo! Finance," http://finance.yahoo.com.

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

[26] D. Vrancic, "Design of Anti-Windup and Bumpless Transfer Protection," Ph.D. dissertation, University of Ljubljana, Slovenia, 1997.

[27] "TPC-DS," http://www.tpc.org/tpcds/tpcds.asp.

[28] N. R. Draper and H. Smith, *Applied Regression Analysis*. Wiley, 1968.

[29] J. L. Devore, *Probability and Statistics for Engineering and the Sciences (6th edition)*. Thomson Learning Inc., 2004.

[30] B. Babcock, M. Datar, and R. Motwani, "Load Shedding for Aggregation Queries over Data Streams," in *International Conference on Data Engineering*, 2004.

[31] N. Tatbul, U. Çetintemel, and S. Zdonik, "Staying FIT: Efficient Load Shedding Techniques for Distributed Stream Processing," in *International Conference on Very Large Data Bases*, 2007.

[32] N. Tatbul, "Load Shedding," in *Encyclopedia of Database Systems*, T. Özsu and L. Liu, Eds. Springer, 2009, pp. 1632–1636.

[33] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms," *Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, vol. 23, no. 1/2, 2002.

[34] C. Lu, Y. Lu, T. Abdelzaher, J. A. Stankovic, and S. Son, "Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 9, pp. 1014–1027, 2006.

[35] Y. Lu, A. Saxena, and T. F. Abdelzaher, "Differentiated Caching Services; A Control-Theoretical Approach," in *Proceedings of the 21st International Conference on Distributed Computing Systems*, 2001, pp. 615–624.

[36] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus, "Using Control Theory to Achieve Service Level Objectives in Performance Management," *Real-Time Systems*, vol. 23, no. 1-2, pp. 127–141, 2002.

[37] W. Kang, S. H. Son, and J. Stankovic, "Design, Implementation, and Evaluation of a QoS-Aware Real-Time Embedded Database," *IEEE Transactions on Computers*, To Appear.

[38] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, "Queueing Model Based Network Server Performance Control," in *Proceedings of the 23rd IEEE Real-Time Systems Symporisum*, 2002, p. 81.

[39] Y. Lu, T. F. Abdelzaher, C. Lu, L. Sha, and X. Liu, "Feedback Control with Queuing-Theoretic Prediction for Relative Delay Guarantees in Web Servers," in *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2003, p. 208.

[40] B. Schroeder, M. Harchol-Balter, A. Iyengar, E. Nahum, and A. Wierman, "How to Determine a Good Multi-Programming Level for External Scheduling," in *Proceedings of the 22nd International Conference on Data Engineering*, 2006, p. 60.

Jisu Oh received her BS degree in computer science from the Kyoungpook National University, MS degree in computer science from University of Minnesota, and PhD degree in computer science from the State University of New York at Binghamton. She is currently working for Samsung Electronics Corp., Korea. Her research interests include real-time data services and wireless sensor networks.

Kyoung-Don Kang is an Associate Professor in the Department of Computer Science at the State University of New York at Binghamton. He received his Ph.D. from the University of Virginia in 2003. His research interests include real-time data services, wireless sensor networks, and security and privacy.