Integrating Proactive and Reactive Approaches for Robust 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*—**
Real-time data services are needed in data-intensive real-time applications such as e-commerce or traffic control. However, it is challenging to support real-time data services, if workloads dynamically change based on the market or traffic status. To enhance the quality of real-time data services even in the presence of dynamic workloads, feedback control theory has been applied. However, a major drawback of feedback control is that it only reacts to performance errors. To improve the robustness of real-time data services, we develop a statistical feed-forward approach that proactively adapts the incoming load, if necessary, to support the desired real-time data service delay. Further, we integrate it with a feedback controller to compensate potential prediction errors and adjust the system behavior in a reactive manner for timely data services. Performance evaluation results acquired in our real-time data service testbed show that our integrated approach considerably reduces the average delay and transient delay fluctuations, while improving throughput compared to the tested baselines including the feed-forward-only and feedback-only approaches.

I. Introduction

In data-intensive real-time applications such as online stock trading, agile manufacturing, and traffic management, it is essential to execute transactions in a timely manner using fresh (temporally consistent) data that reflect the current real-world status. However, it is challenging to achieve the ultimate goal, since workloads may dynamically change based on, for example, the current market or traffic status.

To enhance the quality of real-time data services even in the presence of dynamic workloads, formal feedback control theory [1], [2] has been applied [3], [4], [5]. In these closed-loop approaches, system dynamics are approximated in a piecewise linear manner over a finite operating range of workloads for system modeling [1]. Based on the derived system model, a feedback controller is designed to adjust the system behavior, if necessary, to achieve the desired performance. In our previous work [6], [7], our approaches based on formal feedback control theory implemented in a real database system [8] significantly enhance the data service delay and throughput compared to the tested open-loop and ad hoc feedback approaches.

Unfortunately, feedback control has an inherent limitation; it is essentially a *reactive* approach. A feedback controller must first allow an error to occur before it takes control. As a result, it could lead to a relatively late reaction to bursty workloads or abrupt system state changes, causing poor transient performance such as excessive data service delays. Several previous works including [9], [10], [11] investigate the benefits of seamlessly integrating feed-forward and feedback control techniques to enhance the web server performance. In [9], [10], novel approaches integrating queuing-theory-based predictive approaches with control theoretic techniques are developed to control the Apache web server performance.

The existing feed-forward + feedback approaches for web server performance management is not directly applicable to manage the performance of real-time data services, because queuing theory is not very effective to analyze bursty workloads such as real-time data service workloads [12]. Also, these approaches only consider the queue length to measure the load. In a database system, however, transactions or queries may access considerably different amounts of data. Thus, the queue length is not an accurate metric to measure the database backlog.

In this paper, we develop a feed-forward approach to enhancing the quality of real-time data services and integrate it with a feedback-based approach. Especially, our feed-forward and feedback approaches predict, measure, and adapt the database backlog in terms of the estimated total amount of data to access, i.e., read or write. Therefore, our integrated proactive and reactive approach directly considers database semantics. The main contributions of our work are threefold:

- To enhance the quality of real-time data services, a proactive approach is desired to require no *a priori* knowledge about the underlying workload distribution. Also, it is desired to be independent of a specific statistical workload pattern such as the Poisson distribution [13]. To meet these requirements, we apply a statistical approach independent of the actual distribution of underlying workloads, rather than queuing theory, to enhance the accuracy of predicting the real-time data service delay and accordingly adapt the workload in a proactive manner, if necessary, to support timely data services.
- We integrate the feed-forward approach with a feedback

control scheme for database backlog management [7] to support the desired average and transient delay for data services. Our feed-forward and feedback control approaches adjust the backlog bound via admission control, if necessary, to support the desired average and transient delay for data services in both proactive and reactive ways that are complementary to each other. To our best knowledge, no previous work on real-time data services has been done to integrate a formal feedback-based approach with a feed-forward approach despite the importance.

- We experimentally compare the effectiveness of our feed-forward+feedback scheme to several baselines using real-time data service workloads with different intensity. All our experiments are undertaken in a real-time data service testbed modeling stock trades unlike most existing work on real-time data management relying on simulations [14].

To avoid excessive data service delays in a proactive manner, it is essential to predict the impact of workload dynamics on the system performance. To support high accuracy predictions by observing the current database system status in a fine-grained fashion, we analyze the average delay for accessing a single data item. Generally, a larger (or smaller) average service delay for one data access implies a longer (or shorter) delay for processing the whole transaction or query (i.e., read-only transaction) that includes the data access. Suppose that the estimated service delay for one data item access, called $u(t)$, is known at arbitrary time t. In this case, our proactive admission control scheme is able to adapt the database backlog, if necessary, to support the desired data service delay. For example, suppose that the estimated delay for processing one data item is 0.2 ms and the desired service delay bound for processing a transaction is 2s. In this example, the database should process no more than 10,000 data items to ensure that a transaction is processed in 2s. Thus, our approach is *fine-grained, directly considering database dynamics*.

A key issue is how to predict $u(t)$ despite potential irregularities in workloads. To deal with the irregularity in the underlying distribution of $u(t)$, we take advantage of the Central Limit Theorem (CLT) [13]. The CLT is a powerful statistical tool. It states that the average of a large number of samples tends to approximate a normal distribution *regardless of the underlying statistical distributions* in individual samples [13]. Hence, our feed-forward approach is *not tied to a specific statistical workload model*. Also, our feed-forward approach is *event-driven*. It predicts the backlog bound for real-time data services as soon as a specified number of samples become available; therefore, it does not have to wait for the next feedback control signal computed periodically. (A detailed description of our feed-forward approach is given in Section III.)

To further enhance the performance of real-time data services, we integrate the feed-forward approach with a feedback-based admission control scheme for real-time data services [7]. By integrating them, we create the following benefits for real-time data services:

- Our feed-forward approach substantially reduces the workload burstiness by adapting workloads in a proactive manner, if necessary, to support the desired real-time data service delay. As a result, feedback control becomes more effective too, because the possibility of severe overload significantly exceeding the operating range of workloads used for system modeling and closed-loop design (discussed in Section IV) is reduced via feed-forward control. Although a feed-forward approach may not completely prevent overload conditions, it enhances the robustness of real-time data services.

- As the system status is predicted based on the past delay measurements, feed-forward predictions are not always accurate due to potentially unpredictable workload or system state variations. To compensate possible inaccuracies in predictions, we integrate feed-forward and feedback control techniques. Thus, the integration is mutually beneficial to each other.

- As a result, the integrated approach is more adaptive to dynamic workloads compared to feedback-only approaches for real-time data services such as [4], [3], [5], [6], [7]. This is a key feature needed to improve the quality of real-time data services in dynamic environment.

For performance comparisons, we have conducted extensive experiments in our stock-trading testbed. We extend the Berkeley DB [8] (an open source database from Oracle) to support real-time data services via the feed-forward and feedback schemes. The tested workloads are not only within but also beyond the operating range to explore the system behaviors extensively, unlike most existing work on feedback control of real-time data services [4], [3], [5], [6], [7] that only considers workloads within the operating range. The open-loop approach, i.e., the unmodified Berkeley DB, and feed-forward-only approach based on the CLT fail to support the desired data service delay bound even for relatively light workloads. The feedback-only approach shows stable performance within the operating range; however, the performance degrades noticeably as workloads increase substantially exceeding the operating range. In contrast, the integrated approach closely supports the desired delay bound for data services even for the load that is 8.75 times the load heaviest within the operating range. In the operating range, our integrated approach considerably outperforms the tested baselines in terms of not only service delay but also throughput by proactively smoothing the workload burstiness, while correcting potential inaccuracies in predictions in the closed-loop.

The remainder of this paper is organized as follows. The overall system structure of our database and the overview of our approach are described in Section II. Our feed-forward control approach based on the CLT is discussed in Section III. A description of feedback control design is given in Section IV. Performance evaluation results are described in Section V. Section VI discusses related work. Finally, Section VII concludes the paper and discusses future work.

II. System Overview

In this section, the objective of our QoS-aware data services, the overall system architecture, and a high-level description of our approaches to supporting the desired data service performance are discussed.

A. QoS-Aware Data Service Objectives

In this paper, the service delay s_i of the i^{th} data service request is the sum of the TCP connection delay c_i, queuing delay q_i, and transaction execution delay e_i in the database. In Table I, an exemplar service level agreement (SLA) considered in this paper is shown. In this SLA, the service 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 [15]. We also specify transient performance requirements. 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 S_v reduces to be equal to or less than 2s within the settling time $T_v = 10s$.

For feedback control, the $k^{th}(\geq 1)$ sampling period is the time interval $[(k-1)P, kP)$ and the k^{th} sampling instant is equal to time kP. In this paper, we set the sampling period P = $1s$. Because hundreds of (or more) transactions finish in 1s in our stock trading testbed, performance measurement for $P = 1s$ is reliable. If a considerably larger number of transactions finish accessing a large amounts of data in 1s due to bursty workloads, our feed-forward approach may adapt the backlog bound before the next sampling instant, if necessary, to support timely data services. All the tested approaches, including ours, applying feedback control theory use the same sampling period for fair performance comparisons in Section V.

B. System Architecture

Figure 1 shows the architecture of our QoS-aware data service testbed, called Chronos, built on top of Berkeley DB [8]. Compared to the previous version [7], Chronos is extended by the integrated feed-forward and feedback control schemes.

TABLE I
Desired Performance

Notation	Description	Desired Value
S_t	Desired Service Delay	$2s$
S_v	Delay Overshoot Bound	$2.5s$
T_v	Settling Time	$10s$
P	Sampling Period	$1s$

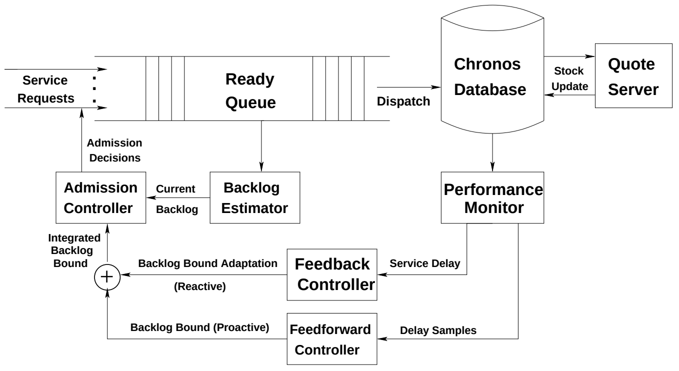


Fig. 1. System Architecture

At time t, the admission controller in Figure I accepts a newly incoming transaction from a client, if the backlog computed by the backlog estimator does not exceed the current backlog bound, $\ell(t)$, after accepting the new request. Otherwise, the database drops the newly arriving requests and returns busy messages to the corresponding clients.

In Figure 1, the feed-forward controller proactively adapts $\ell(t)$ at arbitrary time t to derive $\ell(t+1)$ to be applied to incoming data service requests, if the specified number of samples required for the CLT-based delay prediction becomes available. In addition, the feedback controller periodically adapts the backlog bound to support the desired data service delay. At a sampling instant, the feedback control signal is computed and added to the current backlog bound to compensate the potential inaccuracy of feed-forward control.

The database server schedules accepted requests in an FCFS manner. For concurrency control, we currently apply 2PL (two phase locking). As most existing databases support FCFS and 2PL, our approach is easy to deploy. Also, real-time transaction scheduling and concurrency control schemes [14] are not directly applicable, since most e-commerce applications considered in this paper do not have explicit deadlines for individual transactions.

The database server processes data service requests and periodically updates stock prices received from the stock quote server to support the freshness of stock prices. 3000 stock prices are periodically updated. The update period is in a range $[0.2s, 5s]$ for each stock. We consider periodic temporal data updates, since periodic updates are common in real-time databases to support data temporal consistency [14], [16].

In Chronos, we reserve dedicated threads to update stock prices. These threads for periodic updates are always executed to support the data freshness. On the other hand, admission control is applied to user requests, if necessary, to avoid overload. Further, an admitted user query or transaction has to wait for the next available thread to start execution. Thus, periodic temporal data updates receive higher priorities than user transactions to support the freshness, similar to [16], [14].

C. Estimating Database Backlog

To estimate the database backlog, i.e., the amount of data for the database to process, we leverage our stock trading database schema and semantics of transactions/queries that largely affect the amount of data to process and database performance.

Chronos provides four types of transactions: view-stock, view-portfolio, purchase, and sale for seven tables [7], similar to the TPC-W benchmark [17]. In addition, Chronos supports periodic temporal data updates for real-time data services.[1] For a transaction (or query) T_i from a client, we estimate the amount of data to access by leveraging the information about the database schema and transaction types.

- **view-stock**: To process this query, Chronos 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. By parsing the query, Chronos finds the number of companies n_c specified in the query. Chronos then calculates the amount of data to access for T_i: $n_i = n_c \cdot \{r(\text{STOCKS}) + r(\text{QUOTES})\}$ where $r(x)$ is the average size of a row (i.e., the average number of bytes in a row) in table x.

- **view-portfolio**: Via this query, a client can see certain stock prices in its portfolio. For each stock item in the portfolio, Chronos looks 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 accessed by this query is: $n_i = |portfolio(id)| \cdot \{r(\text{PORTFOLIOS}) + r(\text{QUOTES})\}$ where $|portfolio(id)|$ is the number of stock items in the portfolio owned by the client whose ID is id.

- **purchase**: If a client places a purchase order for a stock item, Chronos first retrieves the current stock price from the QUOTES table. If the purchased stock item was not in the portfolio before the purchase, the stock item and its purchase price are appended to the portfolio. If it is already in the portfolio, Chronos updates the corresponding shares. Hence, the estimated amount of data accessed by a PURCHASE transaction is: $n_i = r(\text{QUOTES}) + (|portfolio(id)| + 1) \cdot r(\text{PORTFOLIOS})$.

- **sale**: A client can sell a subset of its stocks in the portfolio via a sale transaction. To process a sale transaction, Chronos scans the PORTFOLIOS table to look up the stock items belonging to this client's portfolio. Using the stock IDs found in the PORTFOLIOS table, Chronos searches the QUOTES table to find the corresponding stock prices. After that, Chronos updates the client's portfolio in the PORTFOLIOS table to indicate the sale. Thus, the estimated amount of data accessed by a SALE transaction is: $n_i = |portfolio(id)| \cdot r(\text{PORTFOLIOS}) + n_{sell} \cdot r(\text{QUOTES}) + n_{sell} \cdot r(\text{PORTFOLIOS})$ where n_{sell} is the number of stock items to sell.

[1]We do not include temporal data updates for backlog estimation, because temporal data are always updated via resource reservation as discussed in Section II-B.

Let $q(t)$ represent the number of the requests in the ready queue at time t. We compute the database backlog at time t in a fine-grained way:

$$n(t) = \sum_{i=1}^{q(t)} n_i \tag{1}$$

In this paper, 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. For different types of data, such as images, backlog estimation should consider the size of individual data items as well as the number of data to access.

D. Predictive and Reactive Backlog Bound Adaptation

Proactive Adaptation. The feed-forward controller shown in Figure 1 aims to proactively adjust the backlog bound, if necessary, to achieve the target performance specified in Table I. A high-level description of the overall feed-forward control procedure follows.

- If the required number of samples become available at time t, the feed-forward scheme derives the expected delay for processing a single data item, $u(t)$, by applying the CLT in an event-driven fashion (discussed in Section III).

- To support the target service delay S_t described in Table I, the feed-forward approach computes a new backlog bound, which will be used until the next potential update of the backlog bound via either feed-forward or feedback control, using $u(t)$:

$$\ell(t+1) = \frac{S_t}{u(t)} \tag{2}$$

Reactive Backlog Bound Adjustment. The feedback controller models the relation between the database backlog and data service delay, since the delay increases as the backlog increases and vice versa. Based on the system model, the controller is designed to compute how much backlog needs to be adapted to support the desired delay bound (discussed in Section IV). Let $m(k)$ denote the number of the data service requests, i.e., transactions and queries, finished in the k^{th} sampling period. Our feedback-based admission controller works as follows:

- At the k^{th} sampling instant, the feedback controller computes the service delay $s(k) = \sum_{i=1}^{m(k)} s_i/m(k)$ and delay error $e(k) = S_t - s(k)$ where s_i is previously defined in Sections II-A.
- Based on $e(k)$, the feedback controller computes the required backlog bound adjustment $\delta\ell(k)$ at the the k^{th} sampling instant, i.e., time kP.
- Finally, it computes the backlog bound for the next sampling period: $\ell(k+1) = \ell(k) + \delta\ell(k)$. Note that the feed-forward approach may update the new backlog bound before the $(k+1)^{th}$ sampling instant, if new samples become available.

III. Design of Feed-forward Control

Predicting real-time data service delays is important yet challenging, because real-time data service workloads may vary in a bursty manner. To handle the irregularity in the underlying distribution of data processing delays, we apply the Central Limit Theorem (CLT) [13]. According to the CLT, the average of a large number of independent samples, where each sample may have an arbitrary distribution, approximates a normal distribution [13]. More specifically, the average of samples closely approximates a normal distribution when the number of sample N is equal to or bigger than 30. Although taking more samples may increase the accuracy of prediction, we choose to set $N = 30$ to enhance the adaptivity of real-time data services to dynamic workloads.

Also, for each sample, we obtain a sufficient number of delay measurements. In our approach, one sample represents the average delay for accessing one data item computed over 1,000 (or more) recent data accesses. The size of one sample is a tunable parameter. We select 1,000 as the size of a sample to observe a relatively large number of data accesses without waiting excessively to get one sample. In a high performance database for e-commerce, thousands or even more data accesses can be processed per second. In our performance evaluation (Section V), approximately 6,000 – 13,000 data accesses are processed per second.

We use a timer to measure the service delay defined in Section II-A and a counter to count the number of the processed data. Initially, the timer $T = 0$ and counter $C = 0$. When a transaction is finished, we increase the counter C by the number of data accessed by the transaction. If $C \geq 1,000$, we stop the timer and read the total time T spent to process C data items. After that, we compute the average delay for accessing a single data item:

$$u_i = \frac{T}{C} \tag{3}$$

We then reset T and C to zero and repeat the above procedure to obtain 30 samples, i.e., $u_1, u_2, ..., u_{30}$, that are used to predict the delay for servicing one data item by applying the CLT. Whenever the next sample, i.e., u_{31}, becomes available, we move the sliding window by one sample to use $u_2, u_3, ..., u_{31}$ for delay prediction using the most up-to-date set of samples. In this way, we aim to enhance the prediction accuracy in an event-driven manner, while supporting fast adaptivity to dynamic workloads.

The estimated delay for servicing one data access at time t, $u(t)$, is calculated as the p^{th} percentile of the standard normal distribution:

$$P[X \leq u(t)] = p \tag{4}$$

where $u(t)$ is expected to be greater than or equal to $p\%$ of the actual delays for accessing individual data items. Generally, a large percentile value of the distribution yields a good coverage of samples. In this paper, the predicted average delay for processing a single data item is calculated as 99.98 percentile of the standard normal distribution to ensure proper calibration of the estimation. Therefore,

$$P(X \leq u(t)) = \Phi((u(t) - \mu)/\sigma) = 0.9998 \tag{5}$$

where μ, σ, and $\Phi(\cdot)$ are the mean, standard deviation, and density function of the standard normal distribution, respectively [13].

Since the standard normal distribution table [13] gives $\Phi(3.49) = 0.9998$,

$$u(t) = \mu + 3.49 \times \sigma. \tag{6}$$

Using $u(t)$, the newly predicted backlog bound $\ell(t+1)$ is computed according to Eq 2. According to the new backlog bound, the CLT-based approach proactively regulates the incoming workload via database admission control, if necessary, to support the desired data service delay. Notably, our approach can also help system administration. If $u(t)$ continuously increases as t increases, for example, due to the increasing popularity of an e-business, system administrators can acquire and allocate more computational resources to avoid severe degradation in the quality of real-time data services.

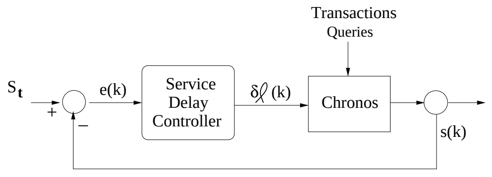


Fig. 2. Data Service Delay Control Loop

IV. Feedback Control Design

We apply well-established feedback control theory [1], [2] to support the desired delay bound together with our feed-forward approach even in the presence of dynamic workloads. To model the relation between the database backlog and delay, we model the data service delay at the k^{th} sampling instant via the service delays and database backlogs measured at the previous p sampling instants. We express the relation in a difference equation in the discrete time domain:

$$s(k) = \sum_{i=1}^{p} \{a_i s(k-i) + b_i \ell(k-i)\} \tag{7}$$

where $p(\geq\ 1)$ is the system order [1], [2]. Using this difference equation, we model database dynamics by considering individual transactions, potentially accessing different numbers of data.

The unknown model coefficients a_i's and b_i's in Eq 7 are derived via system identification (SYSID) [1] to minimize the sum of the squared errors of data service delay estimations based on database backlogs. As SYSID aims to identify the behavior of the controlled database system, Chronos accepts all incoming data service requests. Neither feed-forward nor feedback-based admission control is applied during SYSID. Stock prices are periodically updated as discussed before. In our SYSID procedure, 1200 client threads concurrently send data service requests to the database for one hour. Each client 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. For SYSID, we choose the inter-request time range [0.5s, 3.5s], since the service delay shows a near linear pattern in this area. Each transaction reads or writes 30-50 data items. Therefore, the upper bound of the operating region is: $H_R = 1200 \times 50/0.5 = 120,000$ data access requests/second. Based on H_R, we can observe whether workloads exceed the operating range. Also, we intentionally generate bursty workloads that exceed H_R to test the robustness of real-time data services under stress in Section V. Despite the importance, most existing work on feedback control of real-time data service performance, including [4], [3], [5], [6], [7], does not consider the reliability of real-time data services beyond the operating range.

To analyze the accuracy of SYSID, we use the R^2 metric [1]. A control model is acceptable, if its $R^2 \geq 0.8$ [1]. We derived the first through fourth order models and analyzed the model accuracies via the R^2 metric. We choose the fourth order model that showed the best R^2 value, 0.993:

$$\begin{aligned} s(k) &= 0.1015s(k-1) + 0.2767s(k-2) \\ &\quad +0.4595s(k-3) + 0.0354s(k-4) \\ &\quad +0.01352\ell(k-1) + 0.00032\ell(k-2) \\ &\quad -0.00457\ell(k-3) + 0.00738\ell(k-4) \end{aligned} \tag{8}$$

We derive the transfer function of the open-loop database, i.e., Berkeley DB [8] in this paper, by taking the z-transform [1] of Eq 8:

$$\begin{aligned} B(z) &= \frac{S(z)}{L(z)} \\ &= \frac{0.01352z^3 + 0.00032z^2 - 0.00457z + 0.00738z}{z^4 - 0.1015z^3 - 0.2767z^2 - 0.4595z - 0.0354} \end{aligned} \tag{9}$$

where $S(z)$ is the z-transform of $s(k)$ and $L(z)$ is the z-transform of $\ell(k)$ in Eq 8.

To support the average and transient performance in Table I, we apply an efficient PI (proportional and integral) control law, which can support the stability via I control in addition to P control. We do not use a D (derivative) controller sensitive to noise. At the k^{th} sampling instant, the PI controller computes the control signal $\delta\ell(k)$, i.e., the database backlog adjustment required to support S_t:

$$\delta\ell(k) = \delta\ell(k-1) + K_P[(K_I+1)e(k) - e(k-1)] \tag{10}$$

where the error $e(k) = S_t - s(k)$ at the k^{th} sampling instant as shown in Figure 2. The z-transform of Eq 10 is:

$$P(z) = \frac{\Delta L(z)}{E(z)} = \frac{K_P(K_I+1)[z - 1/(K_I+1)]}{z-1} \tag{11}$$

where $\Delta L(z)$ and $E(z)$ are the z-transform of $\delta\ell(k)$ and $e(k)$, respectively. Given the open loop transfer function in Eq 9 and the transfer function of the PI controller in Eq 11, the transfer function of the closed loop in Figure 2 is: $C(z) = \frac{B(z)P(z)}{1+B(z)P(z)}$ [1].

To support the stability of the closed-loop system, we locate the closed loop poles−the roots of the denominator of $C(z)$−inside the unit circle via the Root Locus method [1], [2]. Hence, our feedback controller can control the desired data service delay to be below the desired bound as long as the workload does not exceed the upper bound of the operating range.

V. Performance Evaluation

In this section, we evaluate our approach and several baselines to observe whether or not they can support the desired performance specified in Table I. We describe our experimental settings followed by the average and transient performance results.

TABLE II
TESTED WORKLOADS

Workload	IRT (200s-600s)	#Threads	Load
W-1	[1s, 1.5s]	1500	$0.625H_R$
W-2	[0.1s, 0.5s]	1500	$6.25H_R$
W-3	[0.1s, 0.5s]	1800	$7.5H_R$
W-4	[0.1s, 0.5s]	2100	$8.75H_R$

TABLE III
TESTED APPROACHES

Approach	Description
Open	Unmodified Berkeley DB [8]
FC	Feedback control of data service delays
CLT	CLT-based admission control
CLT+FC	Integrated approach

A. Experimental Settings

In this set of experiments, we use three identical machines. Each of them has the dual core 1.6GHz CPU and 1GB memory. Every machine runs the 2.6.23 Linux kernel. A Chronos server, clients, and a stock quote server run on each of them, respectively.

For 80% of time, a client thread issues a query about stock prices. For the remaining 20% of time, a client requests a portfolio browsing, purchase, or sale transaction at a time. In fact, most service requests in e-commerce are queries. Thus, this setting is realistic. One experiment runs for 600s. For experimental purposes, we model bursty workloads. At the beginning of one experiment, the inter-request time (IRT) is randomly distributed in [3.5s, 4s]. At 200s, the range of the IRT is suddenly reduced to model bursty workload changes, and stays in the new range until the end of the experiment at 600s as shown in Table II.

To generate different intensity of incoming workloads for evaluation, we use four different workload settings with different IRT ranges during 200s and 600s of an experiment and different numbers of client threads as listed in Table II. Among these tested workloads, W-1 is within the operating range. The highest load generated by W-1 is $0.625H_R$ where H_R is the upper bound of the operating range computed in Section IV. W-2, W-3, and W-4 generate workloads up to $6.25H_R$, $7.5H_R$, and $8.75H_R$, respectively.

To achieve the desired performance, we consider the four approaches shown in Table III. Open, FC, and CLT are the baselines to which the performance of our integrated approach is compared.

- **Open** is the basic Berkeley DB [8].
- **CLT** applies the statistical predictive approach described in Section III without feedback control.

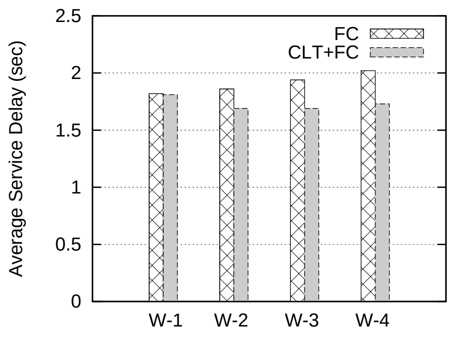


Fig. 3. Average Service Delay

- **FC** is the closed-loop admission control scheme for real-time data services discussed in Section IV. No feed-forward technique is applied.
- **CLT+FC** integrates the CLT and FC.

Each performance data is averaged over 10 runs. We show the performance results observed between 100s and 600s to exclude the database initialization phase.

B. Performance of Open and CLT

Neither Open nor CLT approach is able to support the desired service delay even under the lightest tested workload W-1. Open achieves the $4.64s$ average service delay. CLT decreases the average delay to $3.38s$ by applying the proactive backlog adaptation technique described in Section III. Although the CLT approach itself cannot support the desired service delay, it enhances the service delay by approximately 27% and throughput measured in terms of the total number of the processed data by 18% compared to Open.

CLT itself cannot achieve the desired performance, since it only supports weak convergence [13]; that is, the central limit theorem only states that the average of a large number of samples *approximates* the normal distribution. As OPEN and CLT fail to support the desired delay of data services for even W-1, we omit their performance results in the remainder of this paper.

C. Average Performance Analysis

Figure 3 depicts the average service delays of FC and CLT+FC. In Figure 3, FC supports the desired average service delay of 2s up to W-3. In CLT+FC, the average service delay is controlled to be below the 2s set-point for all the tested workloads. Compared to FC, CLT+FC improves the average service delay up to 12.9%. In Figure 3, the average delay of CLT+FC for W-1 is longer than the delays for the heavier tested workloads, because feedback control dominates within the operating range. As the load increases beyond the

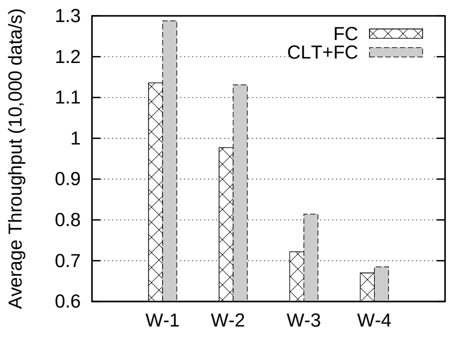


Fig. 4. Average Throughput

operating range, the feed-forward scheme proactively adapts the load more aggressively. As a result, it has more influence on performance for higher workloads.

Figure 4 plots the average throughput. In the figure, the throughput decreases as the load increases for both FC and CLT+FC. However, CLT+FC outperforms the FC in terms of throughput for all the tested workloads. CLT+FC increases the throughput by up to 15.8% compared to FC.

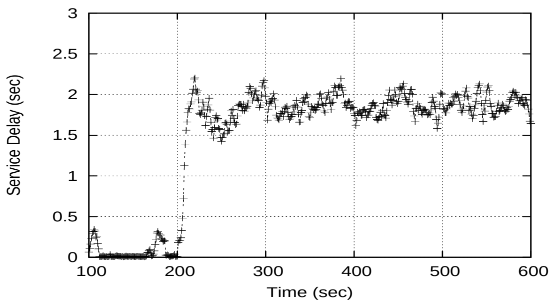
D. Transient Performance Analysis

Figure 5 shows the transient data service delay of FC. FC closely supports the desired data service delay for W-1. In Figure 5, however, it shows more delay oscillations and overshoots for W-2, W-3, and W-4.

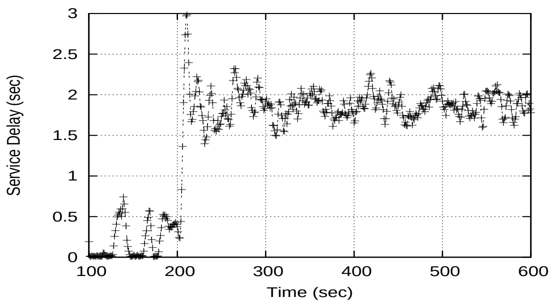
Figure 6 depicts the transient data service delay of CLT+FC. Compared to FC, CLT+FC shows a considerably smaller number of overshoots. Also, the magnitude of overshoots are smaller than that of FC. These results demonstrate that our integrated approach effectively alleviates potentially adverse impacts of bursty workloads on real-time data service performance via proactive backlog adaptation in addition to feedback control of real-time data service performance.

Figure 7 illustrates the number of overshoots occurred between 200s and 600s in our experiments. In FC, the number of overshoots increases dramatically, showing oscillatory delays in Figure 5, as the workload increases. Notably, CLT+FC approach shows substantially fewer overshoots than FC does as shown in Figure 7 for all the tested workloads. In Figures 6 and 7, CLT+FC shows more overshoots for W-1 than the other workloads, because feedback control dominates within the operating range as discussed before. In the future, a more advanced approach will be investigated to reduce overshoots within the operating range too.

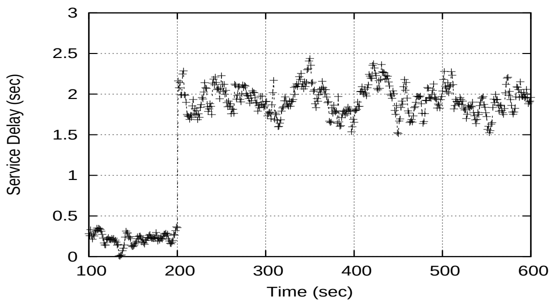
Overall, by seamlessly integrating feed-forward and feedback control techniques, we significantly enhance the robustness of real-time data services under overload compared



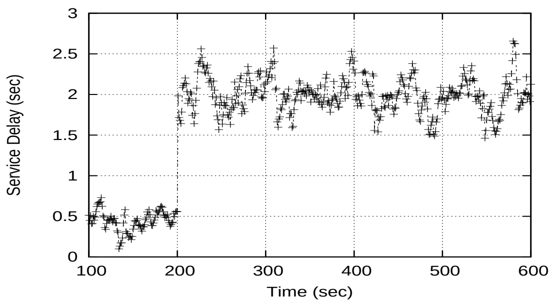
(a) Workload-1



(b) Workload-2

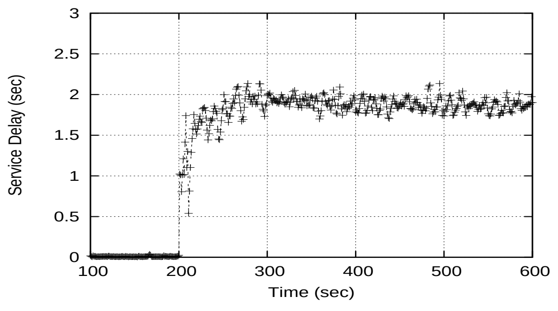


(c) Workload-3

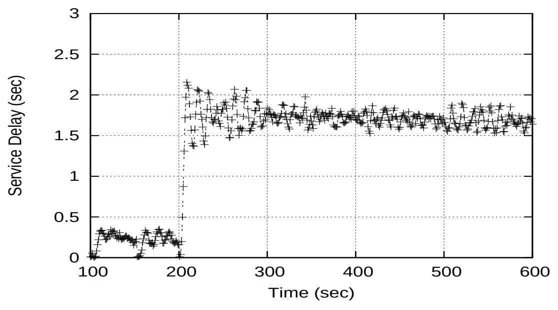


(d) Workload-4

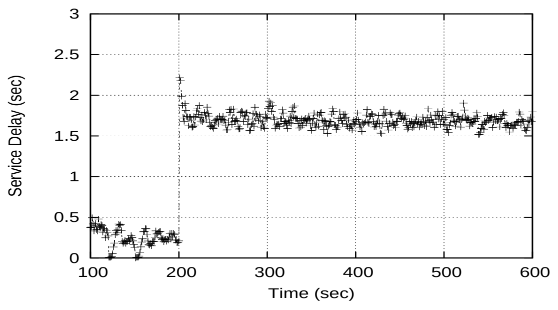
Fig. 5. Transient Service Delay (FC)



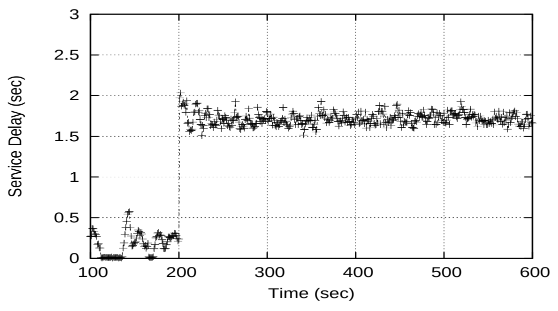
(a) Workload-1



(b) Workload-2



(c) Workload-3



(d) Workload-4

Fig. 6. Transient Service Delay (CLT+FC)

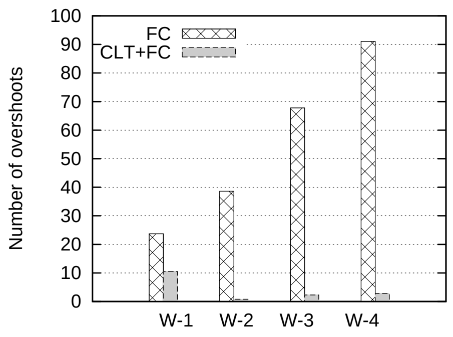


Fig. 7. Number of overshoots

to the tested baselines representing the state of art.

VI. Related Work

Control theory has been applied to manage the performance of computer systems such as a web server [18], [19]. Feedback control is also applied to support real-time data services even in the presence of dynamic workloads [3], [4], [5]. Our previous work [6], [7] applies feedback control theory to manage the performance of a real database system. We are aware of no previous work on real-time data services that applies a predictive approach in addition to formal control theoretic techniques.

Sha et al [9] first investigated the benefits of integrating a queueing-theoretic predictor with a feedback control loop to enhance the web server performance. The queuing-theoretic predictor derives the expected service delay that is then used to dynamically change the service rate in order to fulfill the delay specifications. In [10], the authors use the M/M/1 queueing model to predict the web server delay. The queuing model is integrated with feedback control to manage the web server delay. However, queuing theory is not very effective to model bursty real-time database workloads. It is demonstrated in [12] that the performance of queueing-model-based feedback control degrades in the presence of bursty workloads.

In [11], an improved feed-forward scheme is presented. It replaces the queuing model with a predictor that uses a novel method to predict future delays with no assumption about the statistical properties of the incoming workload. The web server then adapts its resource allocation based on the predicted delay in the closed-loop. Model predictive control technique is applied to manage the CPU utilization in multiprocessor environment [20]. However, these approaches do not consider database-specific issues such as fine-grained data access delay prediction and integrated data service backlog adaptation.

In [21], the authors propose a prediction-based QoS management scheme for periodic queries over dynamic data streams. Their QoS management scheme adjusts the query QoS levels based on online query execution time prediction. However, no control-theoretic approach is applied in their work to compensate potential prediction errors. Further, their prediction scheme is not based on a generally applicable statistical approach such as the CLT. Quality of real-time data service adaptation in addition to admission control in our feed-forward + feedback framework is reserved for future work.

In [22], the authors present a novel approach to achieving statistical QoS guarantees in terms of tardiness in web interactions. However, they consider a number of specific statistical distributions rather than applying a generally applicable concept such as the CLT.

The CLT is applied in [23] to address the problem of estimating the coverage of fault tolerance. A CLT-based statistical approach is also used to predict the execution time for non-deterministic bulk synchronous computations on multiprocessors [24]. However, these approaches do not consider real-time data services. Further, they are not integrated with formal feedback control techniques.

VII. Conclusions and Future Work

Real-time data services are required in data-intensive real-time applications such as e-commerce. However, it is challenging to provide real-time data services due to dynamic workloads. To address the problem, we investigate an effective approach to enhancing the quality of real-time data services and evaluate it as follows:

- We develop a statistical feed-forward approach to adapting workloads in a proactive manner for real-time data services.
- We further improve the quality of real-time data services by integrating feedback control techniques with the proactive approach for database backlog adaptation.
- In the performance evaluation undertaken in a stock trading testbed, our integrated approach substantially reduces the data service delay and delay oscillations, while improving the throughput of real-time data services compared to several tested baselines representing the current state of art.

Considering that there is little prior work on integrated predictive and reactive approaches for real-time data services, our approach can serve as a basis on which more advanced techniques for real-time data services can be built. In the future, we will further investigate more effective approaches for real-time data services in dynamic environments.

References

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

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

[3] M. Amirijoo, J. Hansson, S. Gunnarsson, and S. H. Son, “Experimental Evaluation of Linear Time-Invariant Models for Feedback Performance Control in Real-Time System,” *Real-Time Systems*, vol. 35, no. 3, pp. 209–238, 2007.

[4] 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.

[5] 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.

[6] 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.

[7] K. D. Kang, J. Oh, and Y. Zhou, “Backlog Estimation and Management for Real-Time Data Services,” in *Euromicro Conference on Real-Time Systems*, July 2008.

[8] “Oracle Berkeley DB Product Family,” available at http://www.oracle.com/database/berkeley-db/index.html.

[9] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher, “Queueing Model Based Network Server Performance Control,” in *IEEE Real-Time Systems Symporisum*, December 2002.

[10] Y. Lu, T. F. Abdelzaher, C. Lu, L. Sha, and X. Liu, “Feedback Control with Queueing-Theoretic Prediction for Relative Delay Guarantees in Web Servers,” in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2003.

[11] D. Henriksson, Y. Lu, and T. Abdelzaher, “Improved Prediction for Web Server Delay Control,” in *Euromicro Conference on Real-Time Systems*, 2004.

[12] X. Liu, R. Zheng, J. Heo, Q. Wang, and L. Sha, “Timing Performance Control in Web Server Systems Utilizing Server Internal State Information,” in *Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services*, 2005.

[13] J. L. Devore, *Probability and statistics for engineering and the sciences*, 6th ed. Thomson Learning Inc., 2004.

[14] K. Ramamritham, S. H. Son, and L. C. Dipippo, “Real-Time Databases and Data Services,” *Real-Time Systems Journal*, vol. 28, Nov.-Dec. 2004.

[15] N. Bhatti, A. Bouch, and A. Kuchinsky, “Integrating User-Perceived Quality into Web Server Design,” in *International World Wide Web Conference*, 2000.

[16] M. Xiong and K. Ramamritham, “Deriving deadlines and periods for real-time update transactions,” *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 567–583, 2004.

[17] “Transaction processing performance council,” http://www.tpc.org/.

[18] C. Lu, Y. Lu, T. Abdelzaher, J. 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, Sept. 2006.

[19] 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.

[20] C. Lu, X. Wang, and X. Koutsoukos, “End-to-End Utilization Control in Distributed Real-Time Systems,” in *International Conference on Distributed Computing Systems*, 2004.

[21] Y. Wei, V. Prasad, S. Son, and J. Stankovic, “Prediction-Based QoS Management for Real-Time Data Streams,” in *IEEE Real-Time Systems Symposium*, 2006.

[22] L. Bertini, J. Leite, and D. Mossé, “Statistical QoS Guarantee and Energy-Efficiency in Web Server Clusters,” in *Euromicro Conference on Real-Time Systems*, 2007.

[23] D. Powell, E. Martins, J. Arlat, and Y. Crouzet, “Estimators for Fault Tolerance Coverage Evaluation,” *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 261–274, 1995.

[24] C. Z. Xu, L. Y. Wang, and N. T. Fong, “Stochastic Prediction of Execution Time for Dynamic Bulk Synchronous Computations,” *The Journal of Supercomputing*, vol. 21, no. 1, pp. 91–103, January 2002.