An Adaptive Closed-Loop Approach for Timely Data Services

Dinuni Fernando, Kyoung-Don Kang
State University of New York at Binghamton
{*dferna15, kang*}@*binghamton.edu*

Yan Zhou
Rackspace, Inc.
yan.zhou@rackspace.com

Abstract—**In data-intensive soft real-time applications, e.g., e-commerce, traffic control, and target tracking, a database system needs to process transactions in a timely manner. However, user transactions may suffer from unpredictable large delays when the database system is overloaded due to flash transaction arrivals and transaction aborts/restarts. To address the problem, we design a new adaptive closed-loop method considering database semantics to control the response time to be below a target set-point even when dynamic workloads are given. Our approach continues to update the database model at runtime and re-tunes the response time controller based on the adjusted model, because the relation between the workload and response time may vary in time. Notably, our adaptive control scheme is different from most existing closed-loop methods for real-time database performance management that model the database system and design and tune the controller entirely offline with no online adaptation. The results of the performance evaluation undertaken in a real-time database testbed show that our approach maintains the database response time below the target set-point for most of the time even under steep workload surges, quickly canceling any transient delay overshoot that exceeds the set-point. However, the tested state-of-the-art baselines fail to do it.**

I. Introduction

The demand for timely data services is increasing in a number of data-intensive soft real-time applications, e.g., e-commerce and traffic/weather information services. In these applications, it is desirable for a database system to process user transactions within a desired average response time bound, while reducing short-term transient response time fluctuations.

However, user transactions may suffer from excessive delays when the database system is overloaded. For example, requests for online trade transactions or fastest driving routes may increase in a step-like manner due to unpredictable market conditions or traffic incidents, incurring severe data and resource contention. Due to the excessive database response time under overload, users may miss business opportunities or suffer from increased travel delays.

Classical linear time invariant (LTI) feedback control techniques [1] have been applied to support the desired real-time database (RTDB) performance by continuously adapting the database behavior in the closed-loop [2], [3], [4], [5], [6], [7]. In these approaches, the controlled database system is modeled and the controller is tuned based on the model entirely offline without any online adaptation. However, an LTI closed-loop system may fail to control the database response time to be below the desired target, if the database workload and system behavior largely deviate from the model derived offline.

To address the limitations, we design a novel **adaptive feedback control** approach based on *formal adaptive control theory* [8] to support the desired database response time even when dynamic workloads are given. We model the database system characteristics based on the relation between the *data service delay and database backlog that represents not the queue length but the estimated total amount of data to process*, since different transactions may access different amounts of data. Further, the database response time generally increases as the backlog increases or vice versa.

Notably, in our approach, the controlled **database system model is continuously adjusted online** to reflect the current database system dynamics, because database workloads and the relation between the database backlog and response time may considerably vary in time due to time-varying data/resource conflicts. This approach contrasts to most existing approaches based on *offline* modeling, design, and tuning for closed-loop RTDB performance management, including [2], [3], [4], [5], [6], [7], which do not consider any model update or adjustment at runtime. To the best of our knowledge, no prior work on database performance management has applied adaptive control techniques [8].

Based on the database model continuously updated online, we *systematically adapt the database backlog bound to accept just enough incoming user transactions*, if necessary, to control the response time to be below the target set-point without underutilizing the system. Notably, our adaptive closed-loop system is designed to support the **stability** even in the presence of dynamic workloads and data/resource contention. Further, we analyze the **robustness** of the closed-loop system against dynamic workloads and approximate modeling of the controlled database system, which is very hard to model precisely due to the inherent complexity. By doing the stability and robustness analysis, we analytically ensure that our adaptive feedback control scheme controls the response time to be below a specified target response time bound, while canceling any response time overshoot exceeding the specified bound within a specified settling time even in the presence of dynamic workloads.

For performance evaluation, our approach is actually implemented and evaluated in a database testbed modeling online stock quotes and trades, which contrasts to most existing RTDB work evaluated via high-level simulations without any actual implementation [9]. We thoroughly compare the performance of our approach to those of two state-of-the-art baselines: 1) an unmodified open-loop database system [10] and 2) a closed-loop approach designed and tuned offline by applying non-adaptive (LTI) proportional and integral (PI) control theoretic techniques [1], similar to most existing feedback-based approaches for RTDB performance management [2], [3], [4], [5], [6], [7].

In our performance evaluation, our adaptive control system successfully controls the average database response time to be below the target set-point. Also, it cancels any transient delay overshoot, which exceeds the response time set-point, within the settling time specified at the design time of the adaptive closed-loop system even in the presence of dynamic workloads. However, the baselines representing the state of the art largely fail to support the desired response time for heavy workloads, showing unstable performance. In addition, our adaptive closed-loop approach is **lightweight**. It consumes less than 1% CPU utilization.

The remainder of this paper is organized as follows. The architecture of our adaptive closed-loop real-time database system is described in Section II. Our stock trade database testbed and backlog estimation method are discussed in Section III. A description of our database modeling and adaptive control scheme is given in Section IV. The performance of our approach and baselines is thoroughly evaluated in Section V. Related work is discussed in Section VI. Finally, Section VII concludes the paper and discusses future work.

II. System Architecture

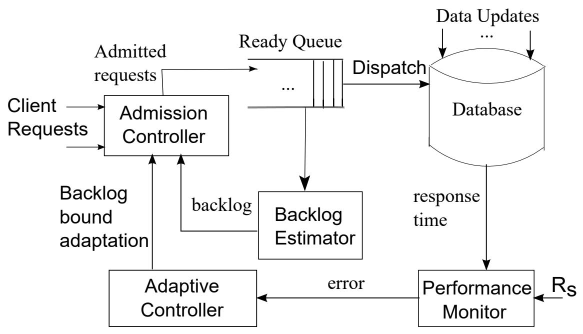

Fig. 1. QoS-Aware Database Architecture

In this section, an overview of our system architecture is given. Database response time management is formulated as an adaptive control problem and the overall structure of our adaptive control scheme is described. Also, the desired target performance is discussed as the control objective considered in this paper.

A. Closed-Loop Database Architecture

Figure 1 shows the adaptive closed-loop database system architecture built atop Berkeley DB [10]. It consists of the database server, performance monitor, adaptive controller, admission controller, and backlog estimator. The database server processes user transactions and periodic updates of temporal data, e.g., stock prices or sensor readings, to support the data freshness.

The performance monitor in Figure 1 measures the average response time of the transactions that finish in each control period and computes the difference between the desired target response time set-point, R_s, and the measured response time to compute the error. Based on the error, the adaptive response time controller adjusts the database backlog bound, if necessary, to support R_s. Generally, the backlog bound is decreased when the system is overloaded or vice versa.

The admission controller admits an arriving user transaction, if it does not expect the backlog bound will be exceeded after accepting it. To this end, the backlog estimator estimates the total amount of data to be accessed by the transactions already in the system and the arriving one. Note that admission control is only applied to user transactions. Periodic update transactions are always scheduled and processed by the system using the dedicated threads to maintain the freshness of data.

In this paper, transactions are scheduled in a FIFO manner. For concurrency control, the two phase locking (2PL) scheme is applied. Since FIFO scheduling and 2PL are supported by most database systems, our approach is easy to deploy.

B. An Overview of Adaptive Database Response Time Control

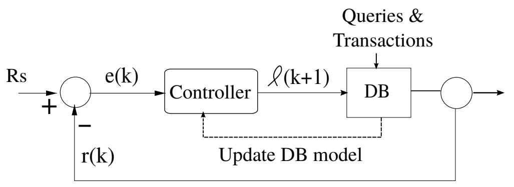

Fig. 2. Adaptive Control Loop for Database Response Time Management

In this paper, the k^{th} $(k \geq 1)$ control point and period indicate the time kT_s and time interval $[(k-1)T_s, kT_s)$, respectively. At the k^{th} control point, the performance monitor in Figure 1 computes the average response time of the transactions that committed in the k^{th} control period:

$$r(k) = \sum_{i=1}^{n(k)} r_i / n(k) \tag{1}$$

where r_i is the response time of transaction i finished in the k^{th} control period and $n(k)$ represents the total number of the transactions completed during the control period. Thus, $r(k)$ is the **output** of the controlled RTDB system at the k^{th} control point.

To support the desired response time R_s even in the presence of dynamic workloads, the response time **error** is calculated at the k^{th} control point as shown in Figure 2:

$$e(k) = R_s - r(k) \tag{2}$$

Based on $e(k)$, the closed-loop system derives the required backlog bound adaptation, $\delta\ell(k)$, and the new backlog bound:

$$\ell(k+1) = \ell(k) + \delta\ell(k) \tag{3}$$

TABLE I
TARGET PERFORMANCE

Notation	Description	Desired Value
R_s	Response time set-point	0.2s
T_v	Settling time	$5s$

where $\ell(k+1)$ is the **control input** provided to the RTDB that will use it to support R_s in the $(k+1)^{th}$ control period. Thus, if $e(k) > 0$, the backlog bound is increased, i.e., $\delta\ell(k) > 0$, or vice versa. (In this paper, $\ell(0)$ is set to a small positive integer when the system is initialized.)

Accurately modeling a database system with complex characteristics is very hard, if at all possible. Hence, we apply formal control theoretic techniques very effective to support the desired performance [8] even when the controlled system model is approximate. At the same time, the database system model is adjusted at each control point as illustrated in Figure 2 to support R_s even in the presence of dynamic workloads and transient system performance.

When a new user transaction arrives at time $t \in [kT_s, (k+1)T_s)$, the estimated total amount of data to be accessed by the transactions already in the system is derived by the backlog estimator in Figure 1:

$$b(t) = \sum_{i=1}^{q(t)} n_i \tag{4}$$

where $q(t)$ is the number of transactions in the ready queue at time t and n_i is the estimated amount of data to be accessed by transaction i.

The admission controller in Figure 1 admits new user transaction j to the system, if $b(t)+n_j \leq \ell(k+1)$ where n_j is the estimated number of data to be accessed by the transaction. Otherwise, the user transaction will be dropped and a system busy message will be returned to the user who can resubmit the request later. For an admission test, only the number of data that will be accessed by the arriving transaction needs to be newly estimated and added to $b(t)$. Thus, computing Eq 4 incurs little overhead upon a request arrival.

C. Target Performance

As shown in Table I, we set the *average response time set-point* $R_s = 0.2$s and the control period $T_s = 0.5$s in this paper. R_s is selected to support the lower bound of the optimal trade interval for U.S. financial markets derived in [11]. Thus, R_s is a realistic real-time database performance goal. When there is a *transient delay overshoot* that exceeds R_s, it needs to be canceled within the specified *settling time*, T_v, to support the reliable transient as well as average performance. In this paper, we require that $T_v \leq 5$s.

Selecting an optimal control period is an open problem [8]. In this paper, we set $T_s = 0.5s$ for both our adaptive controller and the PI controller used as a baseline in Section V.[1] We pick this T_s to frequently monitor the database response time and adapt the backlog bound, if necessary, to closely support R_s. As hundreds of transactions are finished within 0.5s in our database testbed, reliable response time statistics can be acquired at every T_s using Eq 2 too.

[1]The control period is not applicable to the open-loop database baseline that does not rely on a closed-loop system to manage the RTDB performance.

III. DATABASE BACKLOG ESTIMATION

Although our approach is not limited to a specific RTDB application, we build a stock trade database testbed, because there is neither a publicly available RTDB nor a standard benchmark for RTDB research. The database schema and transactions supported by our testbed are similar to the RUBiS online auction benchmark emulating eBay [12] and TPC-E database benchmark that models a brokerage firm [13].

In addition, we support periodic temporal data updates necessary for real-time data services but not supported by RUBiS and TPC-E. In our testbed, there are 3000 temporal data. Each temporal data item, e.g., a stock price or a sensor reading, is updated at a fixed period randomly selected in the range of $[0.2s, 5s]$ during the system initialization to support the data freshness. By doing this, we model frequent updates of temporal data unlike, for example, many financial web sites that only provide a small number of stock price quotes with $15-20$ minute delays [14].

In our stock trade testbed, there are seven tables and four types of transactions: view stocks, view portfolio, purchase, and sales, similar to RUBiS and TPC-E. For each transaction, the backlog estimator in Figure 1 estimates the number of data to be accessed based on the metadata, e.g., the database schema determined at the database design time, as follows.

- *View stock*: To process this query for stock quotes the *stocks* and *quotes* tables that have the stock symbol, full company name, company ID attributes, and current stock prices should be accessed.[2] By parsing the query, the database system finds the number of companies n_c specified in the query. It then computes the estimated amount of data to access to process request i: $n_i = n_c \cdot \{r(\textit{stocks}) + r(\textit{quotes})\}$ 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*: A user issues this query to look up current prices of certain stocks in his/her portfolio and compare them to the purchase prices. For each stock item in the portfolio, the database system looks up the *portfolios* table with the client ID, company ID, purchase price, and shares attributes to find the company IDs necessary to look up the *quotes* table. Thus, the estimated amount of data accessed by this query is: $n_i = |portfolio(id)| \cdot \{r(\textit{portfolios}) + r(\textit{quotes})\}$ where $|portfolio(id)|$ is the number of stock items in the portfolio owned by the customer whose ID is id.
- *Purchase*: A user issues this transaction to purchase a stock item. If the purchased stock item was not in the portfolio before the purchase, the stock item and its

[2]In fact, in our testbed, there are more attributes necessary for online stock quotes and trades. In this paper, we only present the key attributes and tables for the clarity of presentation.

purchase price looked up in the *quotes* table, are added to the portfolio. If it is already in the portfolio, the database system updates the corresponding shares. Hence, the estimated amount of data accessed by a *purchase* transaction is: $n_i = r(\textit{quotes}) + (|portfolio(id)| + 1) \cdot r(\textit{portfolios})$.
- *Sale*: 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 database searches the *quotes* table to look up the corresponding stock prices. After finding the stock prices, the database updates the customer's portfolio in the *portfolios* table to indicate the sale. Thus, the estimated amount of data accessed by a *sale* transaction is: $n_i = |portfolio(id)| \cdot r(\textit{portfolios}) + n_{sell} \cdot r(\textit{quotes}) + n_{sell} \cdot r(\textit{portfolios})$, in which n_{sell} is the number of stock items to sell.

Note that our approach is generally applicable to estimate the backlog in other real-time database applications, e.g., traffic control or target tracking, for the following reasons:

- Usually, the schema is determined at the database design time. Hence, our backlog estimation method can exploit the schema information to estimate the backlog at runtime.
- Transaction semantics for a specific real-time data service application are well known. For timely data services, commonly used transactions, e.g., catalog browsing, purchase, and shopping cart management in e-commerce, are often predefined or canned.

Hence, using these meta data readily available in most database systems, the amount of data to be accessed by a transaction can be estimated.

IV. Database Modeling and Adaptive Response Time Control

In this section, our approach to model the database system and adjust the model at runtime is discussed. To this end, we extend [20] unaware of database semantics by considering the relation between the database backlog and response time as follows.

A. Database System Modeling

To apply control theoretic techniques, it is necessary to model the dynamics of the controlled system, e.g., the database system, in a control theoretic manner. In this paper, the database response time is the *controlled variable* (output). The database backlog bound is the *manipulated variable* (control input) adjusted, if necessary, to support the target response time set-point R_s. Specifically, we require the **convergence** of the database response time to R_s:

$$r(k) = R_s(1 - c^k) \tag{5}$$

where $0 < c < 1$ to ensure the convergence of the response time to R_s with no oscillation.

In this paper, the controlled database system is modeled via a difference equation in the discrete time domain:

$$r(k+1) = \alpha(k)\ell(k) + w(k) \tag{6}$$

where $\alpha(k)$ is an unknown model coefficient adapted at every control point to closely support R_s even given dynamic workloads, $\ell(k)$ is the database backlog bound used in the k^{th} control period, and $w(k)$ is the disturbance, e.g., unexpected data/resource contention.

In this paper, we treat $\alpha(k)$ in Eq 6 as a constant during the k^{th} control period, assuming that it varies slower than the control horizon. Also, accurately predicting $w(k)$ is very hard, if at all possible. Thus, we assume $w(k) \approx 0$ for control modeling purposes, while continuously adjusting the controlled database model in the closed-loop system to compensate for possible modeling errors in terms of the relation between the database backlog and response time. In Section IV-C, we analyze the robustness of our closed-loop system considering the impact of perturbation to analytically verify the validity of our control model.

We convert Eq 6 to the frequency domain by taking the z-transform. In this way, we model the database system algebraically in the frequency domain rather than solving complex partial differential equations in the time domain. A powerful feature of z-transform is that a time delay by i control periods is represented simply by z^{-i} [1]. Thus, by taking z-transform of Eq 6, we get the following equation in the frequency domain:

$$zR(z) = A(z)L(z) \tag{7}$$

where $R(z)$, $A(z)$, and $L(z)$ represent z-transformed $r(k)$, $\alpha(k)$, and $\ell(k)$ in the frequency domain, respectively. As $r(k+1)$ in Eq 6 is the projected response time in the next control period, which is one control period ahead of the other variables, z is multiplied to $R(z)$ in Eq 7.

From Eq 7, we derive the transfer function of the open-loop database system that models the relation between the output from and input to the controlled open-loop database, i.e., the database response time and backlog:

$$P(z) = \frac{R(z)}{L(z)} = \frac{A(z)}{z} \tag{8}$$

We also perform the z-transform of Eq 5 that requires the convergence of the response time to the desired set-point R_s:

$$R(z) = R_s\left[\frac{z}{z-1} - \frac{z}{z-c}\right] = R_s\left[\frac{z(1-c)}{(z-1)(z-c)}\right] \tag{9}$$

If the transfer functions of the open-loop controlled system, i.e., the database system in this paper, and the controller are $X(z)$ and $Y(z)$, the transfer function of the closed-loop system is simply $\frac{X(z)Y(z)}{1+X(z)Y(z)}$ [1], [8]. Thus, to support the target data service delay R_s, we form the closed-loop system transfer function using Eq 8 in the frequency domain:

$$H(z) = \frac{P(z)F(z)}{1 + P(z)F(z)} \tag{10}$$

where $F(z)$ is the transfer function of the response time controller in Figure 2. ($F(z)$ will be derived shortly.)

If our closed-loop system converges to the set-point R_s with zero steady state error, $H(z) = \frac{R(z)}{R_s \cdot \frac{z}{z-1}} = 1$ according to the final value theorem [8]. Therefore, from Eq 9 and Eq 10, we derive that:

$$H(z) = \frac{1-c}{z-c} \tag{11}$$

Thus, the closed-loop pole, i.e., the root of the denominator of $H(z)$, is c. To support the **stability** of the closed-loop system, c should be located within a unit circle [8]. Due to the convergence and stability requirements (Eq 5 and Eq 11), $0 < c < 1$.

We then derive the transfer function of the database response time controller, $F(z)$, to compute the required backlog bound adaptation, $\delta\ell(k)$, if necessary, to support R_s in the $(k+1)^{th}$ control period. To derive $F(z)$, we substitute Eq 8 and Eq 11 into Eq 10:

$$F(z) = \frac{z(1-c)}{A(z)(z-1)} \tag{12}$$

Finally, we take the inverse z-transform of Eq 12 to convert it back to the time domain to compute the control input, i.e., the backlog bound $\ell(k+1)$ that will be used by the RTDB to support R_s in the $(k+1)^{th}$ control period:

$$\ell(k+1) = \ell(k) + \delta\ell(k) = \ell(k) + \frac{1-c}{\alpha(k)}e(k) \tag{13}$$

where $\alpha(k)$ is adjusted at runtime, if necessary, to support R_s as discussed next.

B. Control Gain Adjustment at Runtime

It is important to adjust the backlog bound appropriately, since too high a backlog bound may incur excessive data service delays. On the other hand, too low a bound may result in system underutilization. However, the relation between the database backlog and delay may vary in time. For example, the database response time may increase, if there are severe data conflicts, incurring transaction aborts/restarts.

To address the issue, we dynamically adjust the model coefficient $\alpha(k)$ in Eq 6 to maintain the database response time below R_s based on our database model discussed in Section IV-A. Note that $\alpha(k)$ is also the response time control gain in Eq 13 used to control the database backlog, if necessary, to support the target response time bound R_s. At the k^{th} control point, we estimate the model coefficient as follows:

$$\hat{\alpha}(k) = \lambda \cdot \frac{r(k)}{\ell(k)} + (1-\lambda) \cdot \hat{\alpha}(k-1) \tag{14}$$

where λ is the forgetting factor such that $0 \leq \lambda \leq 1$. If $\lambda = 1$, only the current response time to the backlog ratio is considered to estimate $\hat{\alpha}(k)$. On the other hand, only $\hat{\alpha}(k-1)$ will be used if $\lambda = 0$. The estimated model parameter, $\hat{\alpha}(k)$, replaces the unknown model coefficient, $\alpha(k)$ in Eq 13, to re-tune the adaptive response time controller that computes $\ell(k+1)$ using the new control gain.[3]

In our approach, the estimated model parameter $\hat{\alpha}(k)$ is continuously adjusted; therefore, it is not required to find an optimal model parameter offline. An appropriate value of λ can be derived by solving Eq 14 recursively. For example, it can be tuned to limit the impact of $\frac{r(k)}{\ell(k)}$ on the model parameter to be smaller than 5% after 10 control periods.[4] Note that our approach for adaptive control of the real-time data service delay is **lightweight**, because it only needs to compute Eq 13 and Eq 14 at each control point.

C. Robustness Analysis

To analyze the robustness, we express the exact value of $\alpha(k)$, which is unknown, as follows:

$$\alpha(k) = \hat{\alpha}(k)\psi(k) \tag{15}$$

where $\psi(k)$ is the multiplicative perturbation that indicates the magnitude of the parameter estimation error.

The transfer function of the closed-loop system is derived considering the perturbation impact:

$$H_\psi(z) = \frac{F_\psi(z)P_\psi(z)}{1 + F_\psi(z)P_\psi(z)} = \frac{(1-p)\psi(z)}{z + \psi(z)(1-c) - 1} \tag{16}$$

From this equation and the convergence and stability requirement ($0 < c < 1$), we analyze the robustness of our adaptive closed-loop system against the database model inaccuracy as follows:

$$0 < \psi(k) < \frac{2}{1-c} \tag{17}$$

Thus, choosing a large c close to 1 increases the robustness.

From Eq 5, however, we observe that a large c leads to slow convergence to R_s. Especially, when there is a delay overshoot, it is desirable to minimize the settling time taken to cancel the overshoot and make the response time range between $[(1-\epsilon)R_s, (1+\epsilon)R_s]$ for a small ϵ, e.g., 0.01, as quickly as possible. More specifically, the settling time T_v is:

$$T_v = \frac{\log \epsilon}{\log c} \cdot T_s \tag{18}$$

where T_s is the control period as described before.

From Eq 17 and Eq 18, we observe that there is an inherent *trade-off* between the robustness and settling time; that is, a large c enhances the robustness but increases the settling time or vice versa. In this paper, we pick a relatively small c value to promptly cancel any delay overshoot.

In particular, we set $c = 0.3$. Given that, $0 < \psi(k) < 2.857$ and $T_v = 3.24$s. Thus, our closed-loop system can withstand up to an approximately 285% model coefficient estimation error. Also, any delay overshoot is theoretically expected to

[3]Before the database starts processing user data service requests, $\hat{\alpha}(0)$ is initialized as a small positive real.

[4]A more advanced method, e.g., a Kalman filter, could be used to adapt the model coefficient online by predicting the future database response time based on the history. A thorough investigation is reserved for future work.

TABLE II
STEP-LIKE WORKLOADS

Workload	Average Arrival Rate
W1	1280 → 1560 requests/s (21% ↑)
W2	1280 → 2042 requests/s (60% ↑)
W3	1280 → 2526 requests/s (98% ↑)
W4	1280 → 3200 requests/s (150% ↑)
W5	1280 → 4000 requests/s (213% ↑)

get canceled in 3.24s, meeting the settling time requirement (5s) in Table I.

Overall, our closed-loop system is self-adaptive in that it continuously adjusts the database model and control gain, if necessary, to maintain the response time below R_s.

V. PERFORMANCE EVALUATION

In this section, the performance of our approach and baselines is evaluated to observe whether they can support the desired performance in Table I.

Experimental Settings. The database system, which processes user transactions and periodic temporal data updates, runs in a workstation that has the Intel Core $i7$ 3.6 GHz quad core CPU and 16 GB memory. Client threads run in a separate system with the 3.7 GHz AMD dual core CPU and 8 GB memory. In total, 4800 client threads continuously send user transactions to the database. Each user transaction or query accesses $5-125$ data items. Every machine runs the Linux 3.13.0 kernel. The clients and database are connected through a 1 Gbps Ethernet switch using the TCP protocol. The response time of a real-time data service request sent from a client to the server is the sum of the delays for the network transmission, queuing, and transaction processing in the database system.

The performance of our self-adaptive control (**SAC**) scheme is compared to that of two baselines. First, **Open** is the unmodified Berkeley DB [10] that is a state-of-the-art open source database system provided by Oracle Corp. It simply accepts all user transactions regardless of the current system status. In fact, most database systems do not support admission control.

Second, **PI** extends the Berkeley DB by supporting a feedback-based admission control scheme using a PI controller designed and tuned offline [1], similar to [2], [3], [4], [5], [15], [16], [6], [7]. A PI controller may fail to support the desired performance, if the system behavior largely deviates from the operating range used for offline modeling. For a formal discussion of the offline design and tuning of the closed-loop PI control system, including the operating range set-up, to support R_s, readers are referred to [17] due to space limitations.

To comparatively evaluate the performance of Open, PI, and SAC, the client threads issue queries about stock prices for 60% of the time, because a large fraction of requests are usually quotes in e-commerce, e.g., stock trading, [12], [13]. For the remaining 40% of time, they issue requests for purchase/sale transactions or portfolio browsing in a uniform random manner.

For performance evaluation, we consider several step-like workloads in Table II where the *load is abruptly increased at* $301s$ *and maintained until the end of the experimental run* to model load surges due to, for example, sudden market status changes or traffic incidents. Each experimental run is $900s$ long. In each run, the average user transaction arrival rate is 1280/s between $0s-300s$. At $301s$, the arrival rate is increased suddenly as summarized in the table. For example, for W1 (W5), the average arrival rate is increased from 1280 to 1560 (4000) requests/second at 301s, which is an approximately 21% (213%) increase in the average arrival rate. In this paper, each performance data is the average of 10 runs. 95% confidence intervals are derived for the performance results.

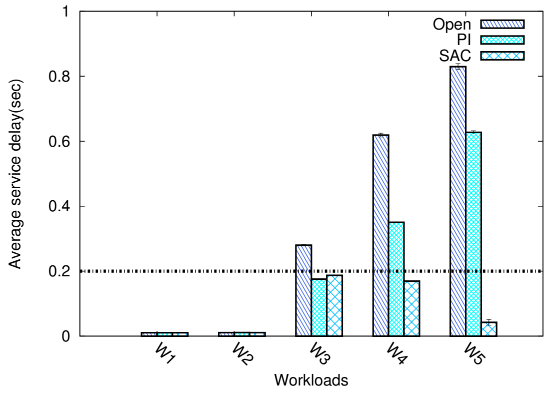

Fig. 3. Average service delay

Average Data Service Delay. In this paper, only the average service delay is presented due to space limitations. For transient performance results, interested readers are referred to [17].

The average data service delays and confidence interval bars of the tested approaches are shown in Figure 3 for the interval $[301s, 900s]$. Before $301s$, the response time of every approach is near zero and much shorter than R_s due to the comparatively low workload in the interval. For W1 and W2, as shown in Figure 3, the average response time of every approach is shorter than $R_s = 0.2s$. However, Open largely fails to support R_s for W3 − W5, because Open is not reactive at all; it simply accepts all transactions regardless of the system status. Generally, PI does a better job than Open does in terms of managing the average response time. The response time of PI does not exceed R_s for W1 − W3. However, PI fails to control the response time to be below R_s for W4 and W5. As PI is designed and tuned offline with no online model and control gain adjustment, it is not adaptive enough to large step workload changes.

In contrast, SAC successfully controls the average response time to be below R_s as shown in Figure 3. Specifically, SAC supports the average response times of $0.18 \pm 0.0007s$, $0.16 \pm 0.0006s$, and $0.04 \pm 0.008s$ for W3, W4, and W5, respectively. SAC adjusts the database model online and adjusts its control gain to make more effective admission control decisions; therefore, it can control the average response time to be below R_s even when the workload increases sharply. We observe that the average response time of SAC decreases as the workload becomes more intense from W3 to W5. This is because its self-adaptive controller becomes more conservative (in terms of admission control) to support R_s for bigger load surges.

VI. Related Work

Classical feedback control techniques [1] have been applied to support the desired performance for real-time data services even in the presence of dynamic workloads [2], [3], [4], [5], [15], [16], [6], [7]. In these approaches, however, the controlled database system is modeled offline without being adjusted at runtime. Also, most of them use an LTI PI controller or linear quadratic regulator designed and tuned offline based on the fixed database model. Therefore, they may fail to support the desired performance, if the workload or system behavior considerably deviates from the offline model. In this paper, we dynamically adjust the database model at each control point and accordingly adapt the database backlog bound, if necessary, to support the desired delay bound for real-time data services.

Control theoretic techniques have been applied to manage the performance of various computational systems, such as real-time systems, web servers, and cloud applications [18], [19], [20]. In [18], model predictive control theory [21], [22] is applied to manage the CPU utilization in a clustered real-time system with no stability analysis. In [20], a novel adaptive control theoretic method is developed for software performance management. Also, the stability and robustness of the adaptive closed-loop system are analyzed. However, they do not consider real-time data service requirements, e.g., transactions, temporal data updates, and database backlog considered in this paper.

VII. Conclusions and Future Work

In data intensive real-time applications, e.g., e-commerce, processing user data service requests in a timely manner is desirable yet challenging due to dynamic workloads and data/resource contention. To address the problem, we design a new lightweight adaptive closed-loop approach to support the desired response time for data service requests even in the presence of dynamic workloads. In our approach, the controlled database system model is adjusted at runtime and the database backlog bound is dynamically adapted, if necessary, to ensure the database response time does not exceed the desired set-point. For performance evaluation, we have actually implemented the state-of-the-art baselines and our approach in a stock trade testbed in contrast to most existing RTDB work. Our approach supports robust average and transient performance in terms of the database response time even when workloads are increased abruptly, whereas the state-of-the-art baselines fail to do it. In the future, we will investigate more advanced approaches to further enhance the efficiency and robustness of real-time data services.

Acknowledgment

We appreciate anonymous reviewers for their help to improve the paper. This work was supported, in part, by NSF grant CNS-1526932.

References

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

[2] M. Amirijoo, N. Chaufette, J. Hansson, S. H. Son, and S. Gunnarsson, "Generalized Performance Management of Multi Class Real-Time Imprecise Data Services," in *IEEE Real-Time Systems Symposium*, 2005.

[3] 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.

[4] 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.

[5] W. Kang, S. Son, and J. Stankovic, "Design, Implementation, and Evaluation of a QoS-Aware Real-Time Embedded Database," *In IEEE Transactions on Computers*, 2012.

[6] J. Oh and K. D. Kang, "A Predictive-Reactive Method for Improving the Robustness of Real-Time Data Services," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 5, pp. 974 – 986, 2013.

[7] Y. Zhou and K. D. Kang, "Deadline Assignment and Feedback Control for Differentiated Real-Time Data Services," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 12, pp. 3245 – 3257, 2015.

[8] K. J. Åström and B. Wittenmark, *Adaptive Control*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1994.

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

[10] "Oracle Berkeley DB," http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html.

[11] D. Fricke and A. Gerig, "Too Fast or Too Slow? Determining the Optimal Speed of Financial Markets," *Social Science Research Network (SSRN)*, 2015.

[12] "RUBiS: Rice University Bidding System," http://rubis.objectweb.org/.

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

[14] "Real Time Quotes," http://www.nasdaq.com/quotes/real-time.aspx.

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

[16] K. D. Kang, Y. Zhou, and J. Oh, "Estimating and Enhancing Real-Time Data Service Delays: Control Theoretic Approaches," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 4, pp. 554 – 567, 2011.

[17] D. Fernando, K. D. Kang, and Y. Zhou, "An Adaptive Closed-Loop Approach for Timely Data Services," State University of New York at Binghamton, Tech. Rep. CS-TR-17-KD01, 2017.

[18] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "DEUCON: Decentralized End-to-End Utilization Control for Distributed Real-Time Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 996–1009, 2007.

[19] 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, pp. 1014–1027, 2006.

[20] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez, "Brownout: Building More Robust Cloud Applications," in *International Conference on Software Engineering*, 2014.

[21] E. Camacho and C. Bordons, *Model Predictive Control*. Springer, 1999.

[22] E. Camponogara, D. Jia, B. Krogh, and S. Talukdar, "Distributed Model Predictive Control," *Control Systems Magazine*, vol. 22, no. 1, pp. 44–52, 2002.