Estimating and Enhancing Real-Time Data Service Delays: Control Theoretic Approaches

Kyoung-Don Kang, Yan Zhou, and Jisu Oh

Abstract—It is essential to process real-time data service requests such as stock quotes and trade transactions in a timely manner using fresh data, which represent the current real world phenomena such as the stock market status. Users may simply leave when the database service delay is excessive. Also, temporally inconsistent data may give an outdated view of the real world status. However, supporting the desired timeliness and freshness is challenging due to dynamic workloads. To address the problem, we present new approaches for (i) database backlog estimation, (ii) fine-grained closed-loop admission control based on the backlog model, and (iii) incoming load smoothing. Our backlog estimation and control theoretic approaches aim to support the desired service delay bound without degrading the data freshness critical for real-time data services. Specifically, we design, implement, and evaluate two feedback controllers based on linear control theory and fuzzy logic control theory to meet the desired service delay. Workload smoothing, under overload, helps the database admit and process more transactions in a timely fashion by probabilistically reducing the burstiness of incoming data service requests. In terms of the data service delay and throughput, our closed-loop admission control and probabilistic load smoothing schemes considerably outperform several baselines in the experiments undertaken in a stock trading database testbed.

Index Terms—Quality of Real-Time Data Service Management, Linear Control Theory, Fuzzy Logic Control Theory.

1 Introduction

In a number of soft real-time applications such as e-commerce and target tracking, it is desirable to process data service requests in a timely manner using fresh data reflecting the current market status or target positions [2]. In e-commerce, for example, most users may simply leave if the service delay exceeds a few seconds. Stale data may give users an unacceptably outdated view of the real world status. Thus, it is desired for a real-time database (RTDB) to support the desired service delay, while periodically updating temporal data such as stock prices [2].

It is challenging to support the desired timeliness and freshness, since database workloads may vary significantly, for example, due to the varying market status. To enhance the quality of soft real-time data services, in this paper, we present several approaches:

- a database backlog estimation technique;
- fine-grained admission control techniques based on the relation between the estimated backlog and service delay. Especially, we develop and compare two admission control schemes by applying linear control theory [3] and fuzzy logic control theory [4], respectively; and
- a hint-based scheme for smoothing incoming workloads.

This work was supported, in part, by US National Science Foundation Grant CNS-0614771.
An early version of this paper [1] was presented at the 20th Euromicro Conference on Real-Time Systems.

- *The authors are with the Department of Computer Science, State University of New York, Binghamton, NY 13902. E-mail: {kang,yzhou,joh}@cs.binghamton.edu.*

To support timely data services, a database needs to be able to estimate the current data service workload and its impact on performance. To quantify the workload, we define the notion of the database backlog to estimate the *amount of data for the database to process*. As it is hard to precisely analyze the backlog due to potential data/resource contention, we estimate the backlog, if any. More specifically, our backlog estimator predicts the amount of data for the database server to actually process by looking up the data service requests in the queue, using the meta data extracted from the database schema and transaction semantics for workload estimation.

Data service workloads may vary in a bursty manner, for example, due to the stock market status. As a result, it is difficult to support the desired timeliness of the service. Further, backlog estimation could be optimistic; a transaction expected to access a certain number of data may get aborted and restarted due to data conflicts. To address this problem, we statistically model the relationship between the *backlog and data service delay* via system identification [3]. This approach is appropriate for database modeling, since the service delay generally increases, if there are more data to process and vice versa. Based on the derived statistical model, we design a feedback-based admission controller based on *linear control theory* [3]. Our closed-loop admission controller computes how much backlog, i.e., the amount of data to process, needs to be reduced when the data service delay exceeds the desired delay bound, e.g., $2s$, and vice versa. Especially, in our approach, the desired average and transient service delays are specified as the service level agreement (SLA) between the database and clients.

In this paper, to support the SLA more closely, we also apply advanced *fuzzy logic control theory* [4] for

fine-grained admission control by directly observing and controlling the nonlinear relation between the backlog and delay. Fuzzy logic control is originally developed to support the desired performance in complex nonlinear systems where high accuracy mathematical modeling of the controlled system, such as a database, is difficult [3], [4]. As fuzzy logic control is not tied to a specific mathematical system model, it is appropriate to manage stochastic database performance potentially affected by disturbances caused by dynamic data/resource contention. Thus, a fuzzy controller is relatively more robust than a linear controller to workload variations and/or underlying hardware platform changes. In this paper, we present a new set of fuzzy rules and a fuzzy logic controller to manage the data service delay via admission control. In our performance evaluation (Section 6), the fuzzy logic controller shows more robustness to workload and platform changes than the linear controller described before.

Linear control theory has previously been applied to manage the real-time database performance [5], [6]; however, most work on feedback control of RTDB performance is analyzed via simulation. In fact, most RTDB work is evaluated based on simulations [2]. Linear control theory is applied to manage a real database performance in a previous work [7]; however, only a gross-grained control model based on the relation between the *queue length* and service delay is employed in [7]. Notably, this model cannot closely capture database dynamics when the transaction size−the amount of data for a transaction to process−considerably varies from transaction to transaction. Compared to them, our approach is based on a fine-grained model for feedback control in a real database. Via database-specific backlog estimation and feedback-based admission control, we can support the desired data service delay without degrading the data freshness unlike most existing work on feedback control of the RTDB performance including [5]–[7]. Further, we are aware of no prior RTDB work that closely supports the desired service delay in a real database system by applying fuzzy logic control.

Moreover, we provide an optional *load smoothing knob* to clients. Individual clients that accept the smoothing option through the SLA *probabilistically reduce the service request rate* under overload by small values predefined in the SLA. The basic idea is that, under overload, an individual client may accept a relatively small additional delay for load smoothing rather than getting rejected by the admission controller. E-commerce clients, for example, may accept additional $0.1s$ delay due to load smoothing predefined in the SLA to increase the chance for their requests to get admitted when the database delay exceeds a few seconds under overload. Especially, individual clients proactively take action rather than waiting until the feedback controller computes a new admission control signal at the next sampling instant. From the system's perspective, flash aggregate workloads can be considerably relieved, if a fraction of clients delay submitting their requests by even a small amount of time. Specifically, we take a simple yet effective approach to load smoothing. The admission controller keeps track of the *request rejection rate* and the database piggybacks the rate to the responses to service requests to minimize the communication overhead. Given the hint, i.e., the current rejection rate, a client delays its next request submission with the probability proportional to the rejection rate. Consequently, resource and data contention can be reduced due to the reduced burstiness of the incoming request traffic. The admission controller can in turn accept more incoming data service requests. Little prior work has considered request traffic smoothing for real-time data services. Further, little work has been done to apply both proactive and feedback control approaches to real-time data services.

For performance comparisons, we have conducted extensive experiments in a stock trading database testbed. In our experiments, feedback-based admission control and probabilistic load smoothing closely support the desired average and transient service delay for bursty workloads. Also, they significantly outperform the tested baselines, which represent the current state of the art, in terms of the average/transient service delay and database throughput. More specifically, our admission controller based on linear control theory reduces the average delay by 109%, 75%, 61% compared to the underlying Berkeley DB [8]−an open source database distributed by Oracle−and the other two baselines, respectively. The load shaping scheme integrated with the feedback-based admission controller enhances the average delay by 156%, 115%, and 98% compared to the same baselines.[1] Our approaches increase the number of the data processed by timely transactions, which finish within the desired delay bound, by $14\% - 53\%$ compared to the baselines. Further, our approaches support reliable transient service delays in the presence of dynamic workloads, whereas the baselines show largely fluctuating transient delays.

In addition, we evaluate the robustness of the linear controller under different workloads and system settings. In this second set of experiments, due to its model-free nature, our fuzzy controller shows the much more reliable average/transient delay and $5\% - 16\%$ higher throughput than the linear controller. These experimental results indicate that fuzzy logic control is a viable approach to managing the timeliness of data services, involving uncertain system behaviors. In addition to the robustness, fuzzy logic control requires less system modeling compared to other control theoretic approaches [3], [4], [9]. Thus, it is relatively simple to design a fuzzy controller. Note that our approaches are lightweight; the linear controller and traffic smoothing scheme only take approximately 1.5% CPU utilization and less than 5KB

1. Note that the delay due to proactive load smoothing via clients is added to the total service delay for fair performance comparisons between the tested approaches that apply and do not apply load smoothing.

of memory. The fuzzy controller is also lightweight. It takes roughly 4% of CPU utilization and less than 1KB of additional memory to store the fuzzy rule-base.

The rest of this paper is organized as follows. The structure of our database and service requirements are described in Section 2. A description of backlog estimation and traffic smoothing is given in Section 3. The design of our linear closed-loop system for admission control is discussed in Section 4. The design of the fuzzy closed-loop system is described in Section 5. Our experimental design and performance evaluation results are described in Section 6. Related work is discussed in Section 7. Finally, Section 8 concludes the paper and discusses future work.

2 Problem Formulation

In this section, the overall data service structure is discussed. The goal of soft real-time data services is set to specify the scope of the work. Also, a high level description of our approaches to supporting the desired data service performance is given.

2.1 Overall Structure

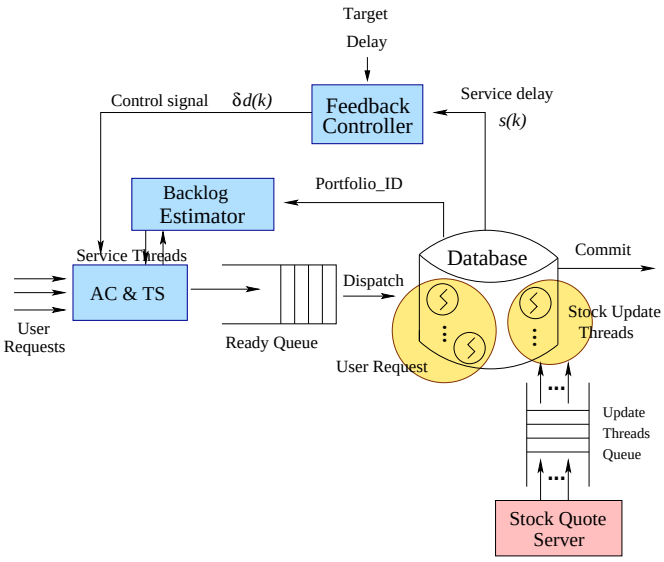


Fig. 1. Chronos Architecture

Figure 1 shows the architecture of our database system, Chronos, built on top of Berkeley DB [8]. It consists of the database backlog estimator, admission controller, traffic smoother, feedback controller, and a database server. The feedback controller is designed based on either linear control theory (Section 4) or fuzzy logic control (Section 5) as discussed before. 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. Chronos periodically updates 3000 stock prices. We consider periodic temporal updates that are commonly used in RTDBs for data temporal consistency [2], [10]. A fixed update period selected in a range $[0.2s, 5s]$ is associated with each stock price. Aperiodic temporal data updates are rarely considered in RTDBs due to the difficulties for defining and maintaining the notion of temporal consistency. A thorough investigation of aperiodic temporal data updates is reserved for future work. As a result, backlog estimation for periodic updates is straightforward. Thus, we focus on estimating the database backlog to service user data service requests in this paper. In Chronos, we reserve dedicated threads for updating stock prices. Also, the dedicated update threads are scheduled in a separate queue ahead of user requests that have to wait for next available threads. Thus, higher priorities are given to temporal data updates to support the data freshness. The database server schedules accepted requests in a FCFS manner, while applying 2PL (two phase locking) for concurrency control. As most databases support FCFS and 2PL, our approach is easy to deploy. Also, most e-commerce applications on which we focus in this paper do not have explicit deadlines for individual transactions or queries. Hence, real-time transaction scheduling and concurrency control schemes are not directly applicable.

TABLE 1
Desired Performance

Notation	Description	Desired Value
S_t	Target Service Delay	$2s$
S_v	Service Delay Overshoot	$2.5s$
T_v	Settling Time	$10s$

2.2 Data Service Objectives

To specify desired performance, a SLA can be specified by an administrator of a soft real-time data service application, such as e-commerce, and agreed with clients. An exemplar SLA considered in this paper consists of the desired data freshness, average/transient service delay, and a predefined range of delays for traffic smoothing (discussed in Section 3). Data freshness, i.e., data temporal validity, is supported by periodically updating temporal data such as stock prices [2]. If the rejection rate is greater than zero, a client probabilistically picks a certain amount of a delay for smoothing in the range predefined in the SLA. Thus, the SLA can be satisfied by supporting the desired average and transient delay specified in Table 1.

As described in Table 1, the average service delay for TCP connection establishment, queuing and data service request processing needs to be shorter than or equal to $S_t = 2s$. We choose this desired bound for real-time data services, because if the average service delay exceeds a few seconds, most of clients are likely to leave. In addition, the transient service delay, i.e., overshoot, is desired to be no longer than 2.5s as shown in Table 1.

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 decays

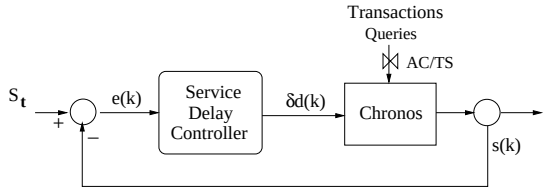


Fig. 2. Data Service Delay Control Loop

within the settling time $T_v = 10s$. Given bursty workloads in Section 6, our feedback-based admission control schemes closely support the desired average/transient delay due to the systematic backlog estimation and controller design performed in Sections $3-5$, while load smoothing further improves the delay and throughput. For feedback control, we set the sampling period P = $1s$ during which several hundreds to thousands of transactions arrive at the database in our testbed.

2.3 Performance Management via Feedback Control

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 in the ready queue in Figure 1, and processing delay e_i inside the database. The $k^{th}(\geq 1)$ sampling period is the time interval $[(k-1)P, kP)$. The k^{th} sampling instant is equal to kP. Further, let $n(k)$ be the number of the data service requests, i.e., transactions and queries, finished in the k^{th} sampling period. Given these definitions, our feedback-based admission controller and load smoother work as follows:

1) At the k^{th} sampling instant, the feedback controller in Figure 2 computes the service delay $s(k) = \sum_{i=1}^{n(k)} s_i/n(k)$ and delay error $e(k) = S_t - s(k)$. Based on $e(k)$, it computes the required backlog bound adjustment $\delta d(k)$.
2) Given $\delta d(k)$, the admission controller updates the database backlog bound to be used in the $(k+1)^{th}$ sampling period:

$$d_t(k) = d_t(k-1) + \delta d(k) \quad (1)$$

where the initial control signal $d_t(0)$ is a small positive integer when the system starts. To manage the backlog bound, we apply the anti-windup technique [11]. If $d_t(k) < 0$ we set $d_t(k) = 0$. To avoid excessive backlogs, we also make $d_t(k) =$ *max-data* if $d_t(k) >$ *max-data* where *max-data* = *max-queue-size* $\cdot$ *max-transaction-size* defined in terms of the amount of data for one transaction to process.[2]
3) In the $(k+1)^{th}$ sampling period, the database accepts a newly incoming transaction as long as the estimated backlog in the ready queue in Figure 1 does not exceed $d_t(k)$ after accepting the new request. Otherwise, the database drops requests at the tail of the queue and returns busy messages to the corresponding clients. Note that we drop requests at the tail rather than the ones at the head of the queue, since clients at the head may experience poor QoS, if their requests are dropped after waiting, for example, several seconds.
4) During the k^{th} sampling period, the database returns the current rejection rate computed by the admission controller to the clients together with data service results or busy messages. Given the information, clients proactively reduce the workload burstiness before the feedback controller computes the next control signal at the k^{th} sampling instant, i.e., time kP.

2. Transactions or queries in soft real-time application such as e-commerce or target tracking are often predefined as well as relatively short and simple compared to transactions in other database applications such as decision support. Further, it is clearly impossible to support a certain delay bound for arbitrarily long transactions.

3 Backlog Estimation and Smoothing

In this section, we describe how to estimate the database backlog by examining the ready queue in Figure 1 and smooth the incoming request traffic under overload.

3.1 Estimating Database Backlog

Chronos provides four types of transactions: view-stock, view-portfolio, purchase, and sale for seven tables [12]. The philosophy behind designing our database schema, transactions, and queries are similar to TPC-W [13], 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. Similarly, RUBiS [14] supports selling, browsing, and bidding for mocked online auctions using seven tables. Further, we add periodic stock price updates for data freshness management critical for real-time data services [2]. To estimate the amount of data to process, we leverage our stock trade database schema and semantics of transactions/queries. Especially, we focus on a subset of the tables directly related to the database backlog estimation. 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, 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. After 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: In this query, a client wants to see certain stock prices in its portfolio. For each stock item in the portfolio, Chronos 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: $n_i = |portfolio(id)| \cdot$

$\{r(\text{PORTFOLIOS}) + r(\text{QUOTES})\}$ where $|portfolio(id)|$ is the number of stock items in the portfolio owned by the client whose ID is id.

purchase: When a client places a purchase order for a stock item, Chronos 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, Chronos updates the corresponding shares. Hence, the estimated amount of data accesses for 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. 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, Chronos scans the PORTFOLIOS table to find 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 for this transaction to process is: $n_i = |portfolio(id)| \cdot r(\text{PORTFOLIOS}) + n_{sell} \cdot r(\text{QUOTES}) + n_{sell} \cdot r(\text{PORTFOLIOS})$.

The database backlog at time t is defined in a fine-grained way:

$$d(t) = \sum_{i=1}^{q(t)} n_i \quad (2)$$

where $q(t)$ is the number of the requests in the ready queue at time t. Suppose a transaction T_i arrives at time t during the $(k+1)^{th}$ sampling period; that is, $kP \leq t < (k+1)P$. T_i is admitted if $d(t) + n_i \leq d_t(k)$ where $d_t(k)$ is computed in Eq 1. 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 [15]. We take this approach, because our stock trading testbed mainly handles structured 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. A view is a virtual table that can be materialized to quickly answer user queries with no database look up [15]. A database can materialize a view by extracting related data from the underlying tables consisting the view. 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.

Overall, our approach is lightweight. We have measured the CPU consumption through the `/proc` interface in Linux. Our backlog estimation procedure described in this section only consumes approximately 0.3% CPU utilization and our feedback control scheme consumes less than 1.2% CPU utilization. Further, they all together consume less than 5KB of memory mainly to store the meta data needed to estimate database backlogs.

3.2 Request Traffic Smoothing

In our approach, traffic smoothing intends to increase the chance for an individual request to get admitted to the system under overload. The admission controller continuously updates the rejection rate $p(k)$ $(0 \leq p(k) \leq 1)$ for transactions arriving in the k^{th} sampling period. The database returns the hint $p(k)$ to the clients as discussed before. Given $p(k)$, a client delays submitting its next service request, if any, with probability $p(k)$. Specifically, a client picks a uniform random number x in [0,1]. If $x < p(k)$, it delays the next request by the time uniformly selected in a range $[t_a, t_b]$ predefined in the SLA.

For analysis, let us assume that the inter-request time between two consecutive data service requests from an arbitrary client uniformly ranges in an interval $[t_1, t_2]$; that is, a client submits $1/0.5(t_1+t_2)$ requests/s in average. If $p(k) > 0$, a request is delayed by the mean time of $0.5(t_a+t_b)$ with probability $p(k)$ due to smoothing. Thus, the client submits $1/0.5(t_1+t_2+p(k)(t_a+t_b))$ requests/s in average. If m clients concurrently submit data service requests, the expected mean reduction of the total arrival rate via smoothing is:

$$E[\gamma] = \frac{2mp(k)(t_a+t_b)}{(t_1+t_2)[t_1+t_2+p(k)(t_a+t_b)]} \text{requests/s.} \quad (3)$$

In the SLA considered in this paper, we set $t_a = 0.1s$ and $t_b = 0.3s$. Hence, under overload, a client's request suffers maximum $0.3s$ extra delay to increase its chance for admission via smoothing. For example, suppose that $t_1 = 0.1s$, $t_2 = 0.5s$, $t_a = 0.1s$, and $t_b = 0.3s$. From Eq 3, it can be computed that, for arbitrary m, the total arrival rate is expected to be reduced by 40% when the rejection rate $p(k) = 1$, while it is expected to be reduced by 25% when $p(k) = 0.5$.

Our smoothing policy has several advantages. First, If $m \gg t_a, t_b, t_1$, and t_2 in Eq 3, our approach can considerably reduce the total arrival rate when $p(k) > 0$. When

overloaded, many clients may submit service requests with short inter-request times. Thus, $E[\gamma]$ in Eq 3 could be large, reducing the burstiness of workloads. As a result, the database server can accept a larger fraction of incoming requests without severely increasing the service delay for individual clients. In Section 6, we show that our smoothing scheme significantly improves the average/transient delay and throughput of real-time data services;

Second, our smoothing technique can enhance the stability of the overall system. By proactively reducing the burstiness, it makes the job of feedback-based admission control easier. In Section 6, the approach integrating traffic smoothing and feedback-based admission control shows the best average/transient performance among the tested approaches.

Further, this approach is distributed requiring no centralized control. It is undertaken by clients based on the hint, $p(k)$, provided by the database. It imposes minimal overheads on the database and clients.

Our traffic smoothing scheme is analogous to network traffic shaping [16]. However, network traffic shaping is implemented by a router, while our approach is voluntarily undertaken by clients based on the hint about the database status. As a result, the burstiness of workloads can be reduced under overload, enhancing the quality of real-time data services in a cost-effective way. Our smoothing approach is driven by the behavior of a data service application supported by Chronos. In contrast, network traffic shaping is oblivious to real-time data service semantics.

4 Service Delay Control via Backlog Adaptation

We apply well-established linear control theory [3], [17] to support the desired delay bound in the presence of dynamic workloads. As the backlog in Eq 2 increases, the service delay increases and vice versa. To model this relation, we express the data service delay at the k^{th} sampling instant based on the n previous service delays and database backlogs in a difference equation expressed in the discrete time domain:

$$s(k) = \sum_{i=1}^{n} \{a_i s(k-i) + b_i d(k-i)\} \quad (4)$$

where $n(\geq 1)$ is the system order [3], [17]. Generally, a higher order model is more accurate but more complex. Using this difference equation, we can model database dynamics considering individual transactions, potentially having different amounts of data to process. Thus, our control model is fine grained.

The unknown model coefficients a_i's and b_i's in Eq 4 are statistically derived via SYSID (system identification) [3], [17] to minimize the sum of the squared errors of data service delay predictions based on database backlogs estimated using Eq 2. As SYSID aims to identify the behavior of the controlled database system,

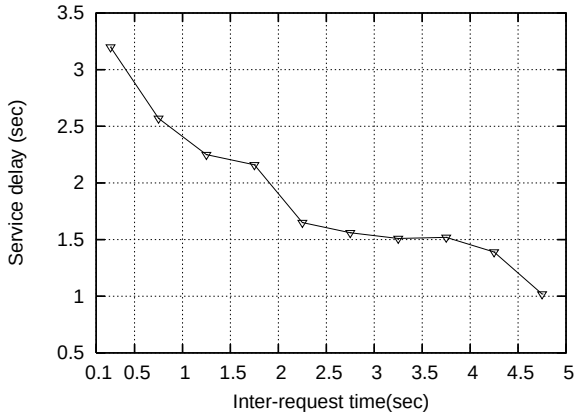


Fig. 3. Inter-request time vs. service delay

Chronos accepts all incoming data service requests. Neither feedback-based admission control nor traffic smoothing is applied during SYSID. Stock prices are periodically updated as discussed before.

For SYSID, 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 processing result, it waits for an inter-request time uniformly selected in a range before sending the next data service request. For SYSID, we choose the inter-request time range $[0.5s, 2.5s]$, since the service delay shows a near linear pattern around $S_t = 2s$ in Figure 3 depicting the inter-request time sub-ranges vs. service delays. Due to our system modeling and controller tuning, the feedback-based admission controller can support the desired delay bound within this operating range. Under overload, smoothing may help the system to re-enter the operating range by reducing the burstiness of incoming data service requests as discussed before.

For accurate SYSID of statistical database dynamics, we use the square root values of the performance data, similar to [18]. To analyze the accuracy of SYSID, we use the R^2 metric [3]. For the second order model, $R^2 = 1-$ variance(service delay prediction error)/variance(actual service delay) = 0.86. A control model is acceptable if its $R^2 \geq 0.8$ [3]. We have found that the first order model is not accurate enough. The third and fourth order models show almost the same R^2. Thus, to model the relation between the database backlog and delay, we favor the second order model, which is less complicated:

$$s(k) = -0.003s(k-1) - 0.104s(k-2) + 0.015d(k-1) + 0.006d(k-2). \quad (5)$$

We derive the transfer function of the open-loop database by taking the z-transform [3] of Eq 5 to algebraically represent the relationship between the database backlog and service delay in the frequency domain:

$$\tau(z) = \frac{S(z)}{D(z)} = \frac{0.015z + 0.006}{z^2 + 0.003z + 0.104} \quad (6)$$

where $S(z)$ is the z-transform of $s(k)$ and $D(z)$ is the z-transform of $d(k)$ in Eq 5.

To support the average and transient performance in Table 1, 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 Derivative controller sensitive to noise. At the k^{th} sampling instant, the PI controller computes the control signal $\delta d(k)$, i.e., the database backlog adjustment needed to support S_t:

$$\delta d(k) = \delta d(k-1) + K_P[(K_I+1)e(k) - e(k-1)] \quad (7)$$

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 7 is:

$$\psi(z) = \frac{\Delta D(z)}{E(z)} = \frac{K_P(K_I+1)[z-1/(K_I+1)]}{z-1} \quad (8)$$

where $\Delta D(z)$ and $E(z)$ are the z-transform of $\delta d(k)$ and $e(k)$, respectively. Given the open loop transfer function in Eq 6 and the transfer function of the PI controller in Eq 8, the transfer function of the closed loop in Figure 2 is: $\eta(z) = \frac{\tau(z)\psi(z)}{1+\tau(z)\psi(z)}$ [3].

To support the stability of the closed-loop system, one needs to locate the closed loop poles−the roots of the denominator of $\eta(z)$−inside the unit circle [3], [17]. To support the stability and performance requirements specified in Table 1, we locate closed loop poles via the Root Locus method supported in MatLab [3], [17]. Specifically, the selected poles are -0.308 and $0.52 \pm 0.106i$. The corresponding $K_P = 2.77$ and $K_I = 5.28$. $\delta d(k)$ computed in the closed-loop is used to compute the backlog bound in Eq 1.

5 Fuzzy Control of Service Delay

A drawback of classical linear control is that it requires significant system modeling. Further, it may or may not be able to support the desired performance when the workload changes beyond the operating range derived by piecewise linear approximation in Section 4. To address this issue, in this section, a fuzzy-logic admission controller is designed to directly model and control the nonlinear relation between the backlog and data service delay. Note that the fuzzy logic controller replaces the linear controller described in Section 4 for feedback control in Figure 1; however, the other system components in Figure 1 are not changed.

5.1 Fuzzy Logic Control Structure

Figure 4 displays the structure of the fuzzy logic controller considered in this paper. To achieve the service delay set-point in Table 1, the fuzzy logic controller computes the service delay error $e(k)$ and change of the error $\delta e(k)$ at the k^{th} sampling instant:

$$e(k) = S_t - s(k) \quad (9)$$

$$\delta e(k) = e(k) - e(k-1). \quad (10)$$

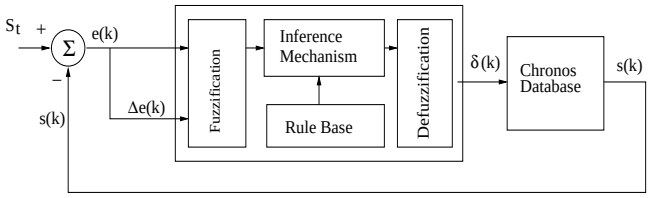


Fig. 4. Fuzzy Logic Control System

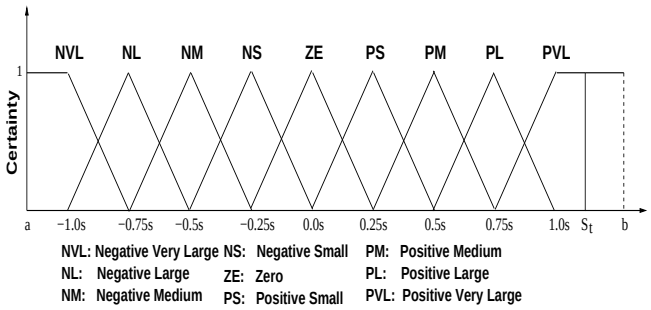


Fig. 5. Membership function for fuzzy input

For fuzzy logic control, one needs to define the universe of discourse (UD) and fuzzy membership functions (MFs). As the service delay ranges $(0, +\infty)$, the UD of $e(k)$ is $(-\infty, S_t)$; therefore $a \rightarrow -\infty$ and $b = S_t$ in Figure 5. As the UD of $\delta e(k)$ is $(-\infty, +\infty)$, $a \rightarrow -\infty$ and $b \rightarrow +\infty$ in Figure 5.

For MFs, we use symmetric triangles of an equal base and 50% overlap with adjacent MFs. By allowing continuous membership unlike traditional set theory, fuzzy logic control aims to deal with uncertainties [4]. As shown in Figure 5, $|e(k)|$ or $|\delta e(k)|$ greater than 1s is considered to be negative or positive very large. In Figure 5, we evenly divide the range $[-1s, 1s]$ into 9 overlapping MFs for fine-grained control of the service delay. Also, the output MFs in Figure 6 are fine grained; the backlog bound increases or decreases by at most 125 data items at one sampling instant. In this way, we intend to avoid large performance fluctuations due to an excessive increase or decrease of the backlog bound for admission control between consecutive sampling instants.

At a sampling instant, the control signal is generated through fuzzification, inference, and defuzzification as shown in Figure 4. In **Step 1**, the **fuzzification** interface in Figure 4 converts $e(k)$ and $\delta e(k)$ to linguistic values

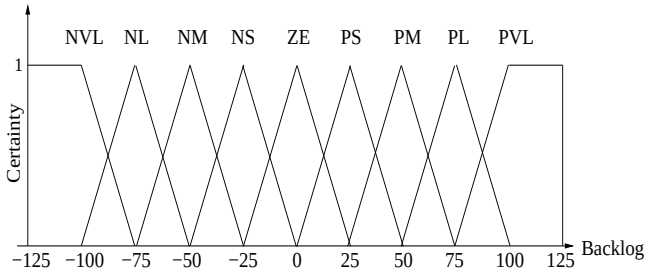


Fig. 6. Membership function for fuzzy output

such as NL (Negative Large) or PM (Positive Medium). The certainty of $e(k)$ or $\delta e(k)$ to be associated with (a) corresponding linguistic value(s) is quantified by the MF(s) for the fuzzy input depicted in Figure 5. For example, suppose that the measured service delay at the k^{th} sampling instant $s(k)$ is 2.2s and the service delay at the previous sampling instant $s(k-1) = 1.8s$. According to Eq 9 and Eq 10, $e(k) = -0.2s$ and $\delta e(k) = -0.4s$ in this example. The error $e(k) = -0.2s$ is linguistically Negative Small (NS) with the certainty $\mu_{NS}(-0.2s) = 0.8$ and Zero (ZE) with the certainty $\mu_{ZE}(-0.2s) = 0.2$ according to the MFs in Figure 5. $\delta e(k) = -0.4s$ is Negative Medium (NM) with the certainty 0.6 and NS with the certainty 0.4. To express key ideas for fuzzy logic control, we will continue to use this example in the remainder of this subsection.

In **Step 2**, given the fuzzified $e(k)$ and $\delta e(k)$, the **inference** mechanism in Figure 4 looks up the fuzzy rule-base (shown in Table 2) to find the corresponding linguistic control signal(s). The rule-base consists of a set of IF-THEN rules that directly map fuzzy inputs to fuzzy outputs. These IF-THEN rules basically indicate that IF the system behaves in a certain manner in terms of the measured performance, THEN certain control actions should be taken to adjust the workload, if necessary, to support the desired service delay. By referring to the rule-base in Table 2, the inference engine finds that four rules, i.e., $rule(NS, NM) = NL$, $rule(NS, NS) = NM$, $rule(ZE, NM) = NM$, and $rule(ZE, NS) = NS$, are relevant.

To compute the certainty of the applicability of the corresponding IF precedent THEN consequent rule(s), we take the minimum of the certainty values of $e(k)$ and $\delta e(k)$. This is a logical approach, because the certainty of the consequent cannot be higher than the certainty of a precedent [4]. Thus, for the $rule(e(k), \delta e(k))$ in the rule-base, the certainty $\mu(e(k), \delta e(k)) = \mu(NS, NM) = min\{0.8, 0.6\} = 0.6$, $\mu(NS, NS) = min\{0.8, 0.4\} = 0.4$, $\mu(ZE, NM) = min\{0.2, 0.6\} = 0.2$ and $\mu(ZE, NS) = min\{0.2, 0.4\} = 0.2$.

In **Step 3**, the **defuzzification** interface in Figure 4 converts the linguistic conclusions reached by the inference mechanism into the actual control signal that is expressed as a single real number to be used by the admission controller. Specifically, the defuzzification interface derives $\delta d(k)$, which is the backlog bound adjustment needed to achieve the desired service delay in Eq 1. To do this, we apply a popular defuzzification method called *center of gravity method* [4]. Let $\mu(i, j)$ denote the certainty of the $rule(i, j)$ computed in the previous step and $c(i, j)$ denote the center of the MF for the consequent of the $rule(i, j)$. The center of a triangular MF in Figure 6 is the horizontal axis value at the middle of the corresponding triangle's base. For example, the center of NM is -50. From these, the fuzzy control output is:

$$\delta d(k) = \frac{\sum_{i,j} c(i,j) \cdot \mu(i,j)}{\sum_{i,j} \mu(i,j)} \quad (11)$$

By applying Eq 11 to the previous example, the defuzzified control signal $\delta d(k) = (-75 \cdot 0.6 + -50 \cdot 0.4 + -50 \cdot 0.2 + -25 \cdot 0.2)/(0.6 + 0.4 + 0.2 + 0.2) = -57.15$.

Finally, the admission controller **updates the database backlog bound** to be used in the $(k+1)^{th}$ sampling period using Eq 1. In the $(k+1)^{th}$ sampling period, the admission controller only accepts a newly incoming transaction, if the estimated backlog does not exceed the backlog bound $d_t(k+1)$ after accepting it. Otherwise, the database drops this request. The whole procedure described in this subsection is repeated at every sampling instant to support the desired service delay S_t. Notably, this backlog bound update and admission control policy are the same as the corresponding part in the linear control approach described in Section 4. The difference lies in the method to compute the control signal, i.e., the backlog adjustment.

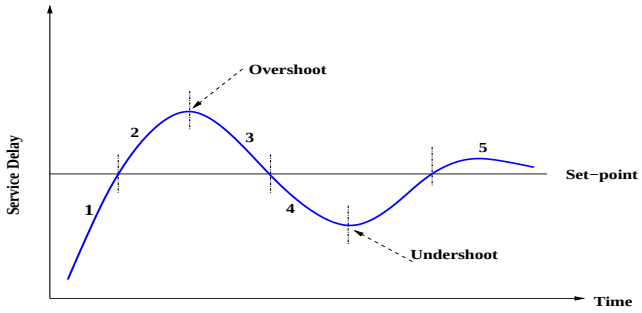


Fig. 7. Fuzzy Control Characteristics

5.2 Key Ideas for Rule-Base Design

In this paper, we use 9 different linguistic values to characterize both the fuzzy input and output as shown in Figure 5. Therefore, the rule base includes $81(9 \times 9)$ rules described in Table 2. As shown in Figure 7, there are five zones that characterize potential data service variations, from which we derive the rule-base.

Zone 1: In this zone, the measured service delay is smaller than the set-point, yet it comes closer to the set-point. In general, we increase the backlog bound to enhance the database throughput. At the same time, however, care should be taken to avoid a potential overshoot that can happen in the next sampling period due to the excessively increased load. To avoid this problem, we take many different control actions based on different fuzzy inputs as shown in Table 2. For example, suppose that $e(k)$ is PVL. If $\delta e(k)$ is NS in this case, the control action needed to increase the load is not fast enough; therefore, we apply PL control signal to the database. On the other hand, if $\delta e(k)$ is NVL, ZE is applied to the system, because the system is quickly correcting its own behavior. As another example, assume that $e(k)$ is PS but $\delta e(k)$ is NVL. In this case, the controller is too

TABLE 2
Rule Base

$e/\Delta e$	NVL	NL	NM	NS	ZE	PS	PM	PL	PVL
NVL	NVL	NVL	NVL	NL	NL	NL	NM	NS	ZE
NL	NVL	NVL	NL	NL	NL	NM	NS	ZE	PS
NM	NVL	NL	NL	NL	NM	NS	ZE	PS	PM
NS	NL	NL	NL	NM	NS	ZE	PS	PM	PL
ZE	NL	NL	NM	NS	ZE	PS	PM	PL	PL
PS	NL	NM	NS	ZE	PS	PM	PL	PL	PVL
PM	NM	NS	ZE	PS	PM	PL	PL	PVL	PVL
PL	NS	ZE	PS	PM	PL	PL	PVL	PVL	PVL
PVL	ZE	PS	PM	PL	PL	PVL	PVL	PVL	PVL

aggressive, increasing the load too fast. Thus, we apply NL to avoid a potential overshoot by decreasing the load.

Zone 2: In this zone, the actual service delay is larger than the set-point and it is further increasing. The controller must reduce the load to reverse the current trend by applying a relatively large negative control signal.

Zone 3: In this zone, the actual service delay is higher than the set-point, yet it shows the trend to come closer to the set-point. The controller applies a negative signal to trigger the service delay decrease when $e(k)$ is still negative large. If $e(k)$ is decreasing sharply or showing a trend to converge to the set-point, we apply relatively small negative signal to avoid a potential undershoot where the system is underutilized. Similar to Zone 1, careful control is required to avoid an undershoot by not decreasing the load excessively to support the desired service delay. For instance, suppose that $e(k)$ is NVL and $\delta e(k)$ is PS. In this case, the service delay is much higher than the set-point but the workload does not decrease fast enough; therefore, we apply NL to expedite the convergence to the set-point. As another example, assume that $e(k)$ is NS and $\delta e(k)$ is PM. Since the service delay is sightly higher than the set-point but still about to decrease significantly, we apply PS to avoid a potential undershoot.

Zone 4: The measured service delay is lower than the set-point and it is further decreasing. Thus, the controller increases the backlog bound to accept more requests to avoid further undershoots.

Zone 5: $|e(k)| \leq \sigma$ and $|\delta e(k)| \leq \sigma$ where σ is a predefined small positive real number. The service delay converges to the set-point and $|e(k)|$ and $|\delta e(k)|$ are small. Hence, the value of backlog adjustment $|\delta d(k)|$ is small (or zero) to expedite the convergence to the set-point, while reducing the magnitude of the potential oscillations.

6 Performance Evaluation

In this section, we evaluate our approach and several baselines to observe whether or not they can support the desired performance in Table 1. A description of experimental settings is followed by the average and transient performance results. The performance of the linear controller and traffic smoothing scheme is compared to several baselines in Section 6.1. The performance of the fuzzy control approaches is compared to linear control approaches in Section 6.2.

6.1 Baselines and Linear Control Approaches

Experimental Settings. A Chronos server runs in a laptop with the 1.66 GHz dual core CPU and 1 GB memory. Clients run in a desktop PC that has the 3 GHz CPU and 2GB memory. In total, 1200 client threads continuously send service requests to the Chronos server. The stock quote server runs in a desktop PC that has the same CPU and 4 GB memory. Every machine runs the 2.6.15 Linux kernel. The clients, stock quote server, and database sever are connected through a 1 Gbps Ethernet switch using the TCP/IP protocol.

For 60% of time, a client thread issues a query about stock prices, because a large fraction of requests can be stock quotes in stock trading. For the remaining 40% of time, a client uniformly requests a portfolio browsing, purchase, or sale transaction at a time. Considering data access skews incurring hot-spots in a database is reserved as a future work. As one of the first work on real implementation and evaluation of feedback-based techniques for managing real-time data service performance, we mainly focus on evaluating the basic effectiveness of our approaches in this paper.

One experiment runs for $900s$. At the beginning of an experiment, the inter-request time is uniformly distributed in [$3.5s$, $4s$]. At $300s$, the range of the inter-request time is suddenly reduced to [$0.1s$, $0.5s$] to model bursty workload changes; that is, the total 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 $900s$. This is considerably more bursty than our previous testbed workloads [7]. Also, most existing RTDB work is not evaluated in a real database system using bursty workloads [2], [7].

Tested Approaches. For performance comparisons, we consider the five approaches shown in Table 3. *Open* is the basic Berkeley DB [8] without any special performance control facility. Hence, it represents a state-of-

TABLE 3
Tested Approaches

Open	Pure Berkeley DB [8]
AC	Ad-hoc admission control
FC-Q	Feedback control (FC)−queue size vs. delay
FC-D	FC−database backlog vs. delay
FC-TS	FC of database backlog & traffic smoothing

the-art database system. For performance comparisons, all the other tested approaches are implemented atop Berkeley DB.

AC applies admission control to incoming transactions in proportion to the service delay error under overload.

FC-Q models the relation between the queue size and response time, similar to [7]. The controller in FC-Q computes the required *queue length* adaptation to support the desired response time. The control signal for queue size adaptation is enforced by applying admission control to incoming transactions. Since the workloads used in this paper are different from−more bursty than−the workloads used in [7], we re-tuned FC-Q's controller based on the relation between the queue length and delay to support the performance requirements described in Table 1. Open, AC, and FC-Q are the baselines to which the performance of our approaches is compared.

FC-D is the closed-loop admission control scheme based on linear control theory (Section 4).

FC-TS extends FC-D by supporting traffic smoothing in addition to closed-loop admission control under overload. For fair comparisons, we add the delay due to traffic smoothing, if any, to the service delay of FC-TS as discussed before. Specifically, we set the range of the delay for traffic smoothing to [$0.1s$, $0.3s$] as described in Section 3.2.

Each performance data is the average of 10 runs using different random seeds. 90% confidence intervals are also derived. We show the performance results observed between $100s$ and $900s$ to exclude the database initialization phase, involving initial housekeeping chores such as database schema and data structure initialization.

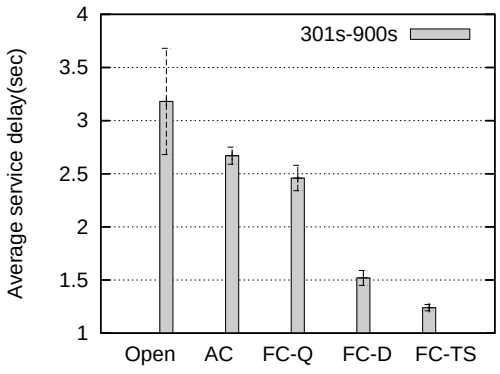


Fig. 8. Average service delay

Average Data Service Delay. Figure 8 shows the average service delay for each tested approach. In the interval [$100s$, $300s$], all the tested approaches can support the desired delay bound $S_t = 2s$ (Table 1) due to the relatively low workload. However, only FC-D and FC-TS can support the delay bound in [$301s$, $900s$] as shown in the figure. FC-D achieves the $1.52 \pm 0.07s$ average service delay via closed-loop admission control based on the database backlog model. FC-TS further reduces the average delay to $1.24 \pm 0.03s$ via traffic smoothing in addition to feedback control. Open fails to support S_t by accepting all incoming requests, causing system overloads. AC fails to support S_t due to its ad hoc admission control. From this, we observe that systematic feedback control is necessary. The average delay of FC-Q is shorter than Open and AC. However, FC-Q's control model is only based on the queue length vs. delay, which is coarse grained. Thus, it cannot support the delay bound under overload. FC-Q was only able to support the delay bound for dynamic workloads, when the data freshness was degraded too [7].

Transient Service Delay. Figure 9 plots the transient service delay measured at every $1s$ sampling period. FC-D and FC-TS do not suffer any delay overshoot in the [$100s$, $300s$] interval. In contrast, Open, AC, and FC-Q suffer overshoots even in this range. This implies that the database can be transiently overloaded due to dynamic data/resource contention for even moderate workloads. Hence, fine-grained, database-specific workload estimation, feedback control, and traffic smoothing are desirable. In Figure 9, after the bursty workload increase at $300s$, FC-D significantly outperforms the baselines in terms of the transient service delay. FC-D supports S_t for most of the time despite bursty workloads. There are a few overshoots after $300s$; however, most of them are smaller than $S_v = 2.5s$ and they decay within the desired settling time $T_v = 10s$ specified in Table 1. The largest overshoot is $4.13s$ at $542s$, but it decays in $3s$. FC-TS supports even better transient performance: Its highest delay overshoot is $2.88s$ at $620s$ and it decays in only $3s$. FC-Q performs better than AC and Open, but it largely violates the desired transient performance specified in Table 1. Moreover, FC-D and FC-TS process much more data than the baselines do as discussed next.

Data Service Throughput. Figure 10 shows the average number of the data processed by the committed transactions, which is *normalized* to the corresponding number for FC-D. There are two bars for each tested approach. The right bar is the total number of data processed during $900s$, while the left bar indicates the number of data processed by the timely transactions or queries completed within S_t. We call it the number of *timely* data. As shown in the figure, FC-D and FC-TS increase the total number of processed data and the number of timely data by approximately $9\% - 48\%$ and $14\% - 53\%$ compared to FC-Q, AC, and Open.

FC-D has processed more than 18,680,000 data accesses in $900s$. Out of the total number of the data processed by the committed transactions, more than

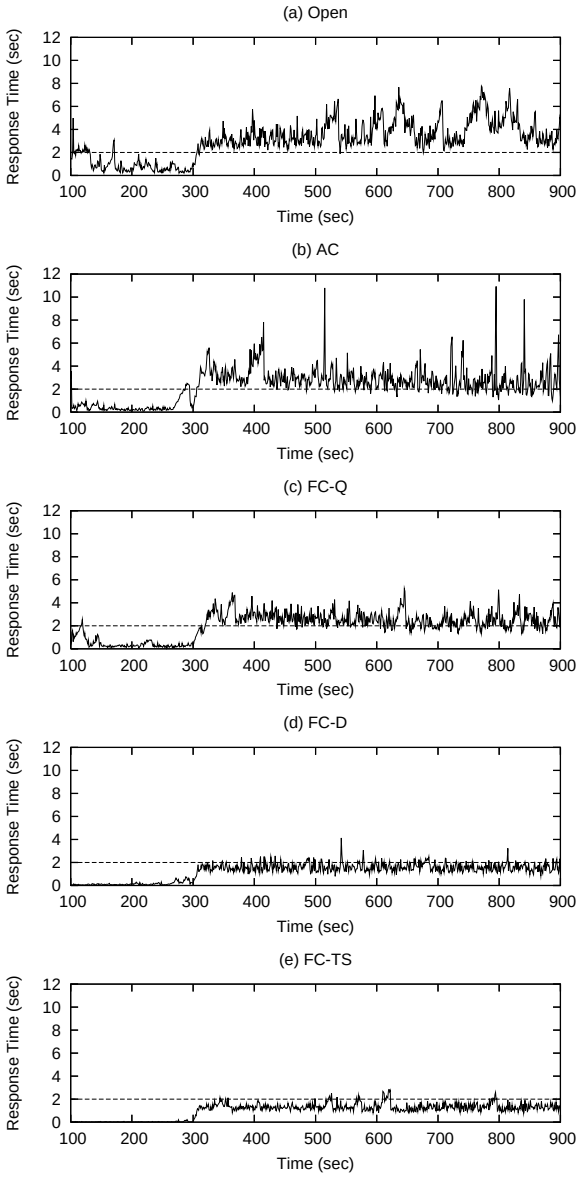


Fig. 9. Transient service delay

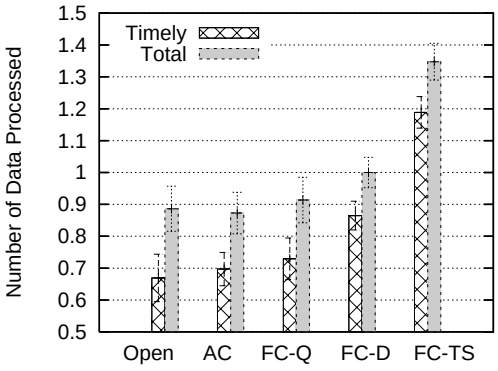


Fig. 10. Normalized average data accesses

16,150,000 data accesses are served by the transactions committed within S_t, which is approximately 86% of the total data processed. By managing the database performance based on the amount of data to process, FC-D effectively avoids excessive service delays. As a result, it processes more data in a timely fashion than the baseline approaches. By applying smoothing in addition to feedback-based admission control, FC-TS processes 25,175,939 data in $900s$. Approximately 88% of them are processed in a timely manner.

We have also observed that our approach significantly improves the transient database throughput compared to the baselines. However, we do not include the performance evaluation results due to space limitations. In summary, FC-D closely supports the desired delay specified in Table 1, while considerably enhancing the throughput. FC-TS further improves the average and transient service delay. Its throughput is significantly higher than the other approaches. Since the burstiness of workloads and resulting data/resource contention can be considerably relieved by traffic smoothing, more transactions can be admitted and processed within the desired delay bound. On the other hand, individual clients do not suffer excessive extra delays due to traffic smoothing, because the delay due to traffic smoothing, if any, is bounded by the predefined maximum as discussed in Section 3. Overall, our database-specific backlog estimation and service quality management schemes are very effective.

6.2 Performance Comparisons between Linear and Fuzzy Logic Control

In this subsection, we compare the performance of the linear and fuzzy logic controllers under a different system and workload setting. For performance comparisons, we consider four approaches shown in Table 4 where FZ-D is the fuzzy-logic-based admission control scheme described in Section 5. Also, FZ-TS extends FZ-D by applying workload smoothing. We do not consider the baselines, as they failed to support the desired performance in the previous experiments.

Experimental Settings. In this set of experiments, we use three identical desktop PCs. Each of them has the dual core 1.6GHz CPU and 1GB memory. Every machine runs the 2.6.23 Linux kernel. Chronos server, clients, and stock quote server run on each of them respectively. Note that, in this subsection, the quote server has 3GB less memory than it did in the previous subsection. As a result, the database server can be considerably delayed to wait for stock price updates, while serving concurrent clients. Thus, the delays in Figure 11 are generally longer than the delays in Figure 3.

Different machines are used in this set of experiments, since the machines used in Section 6.1 became unavailable due to maintenance. Also, using different machines, we can measure the robustness of the tested approaches across different platforms.

For fair performance comparisons, we have re-tuned the PI controller used by FC-D and FC-TS for the new

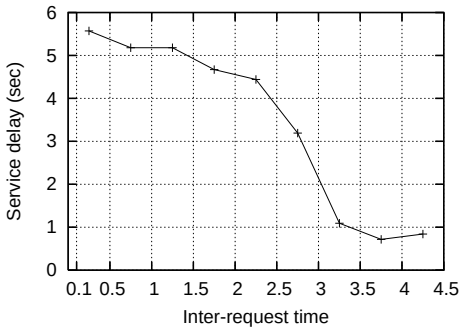


Fig. 11. Inter-request time vs. service delay (New setting)

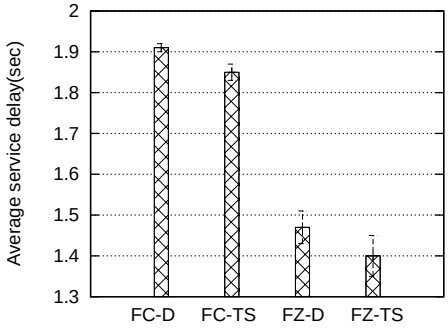


Fig. 12. Average Service Delay (1200 clients)

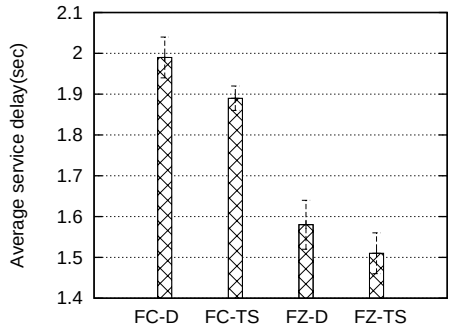


Fig. 13. Average Service Delay (1500 clients)

workload and system settings following the procedure described in Section 4. For tuning, 1200 clients submit service requests, similar to Section 4. The operating range selected for SYSID and tuning is the inter-request time range of [2s, 4s] showing a near linear relation between the service request rate and delay in Figure 11.

For 80% of time, a client thread issues a query about stock prices. For the remaining 20% of time, a client uniformly requests a portfolio browsing, purchase, or sale transaction at a time. One experiment runs for 600s. At the beginning of an experiment, the inter-request time is uniformly distributed in [3.5s, 4s]. At 200s, the range of the inter-request time is suddenly reduced to [1s, 1.5s] to model bursty workload changes. Each performance data is the average of 10 runs using different random seeds.

To further evaluate the robustness of the PI and fuzzy controllers, we perform the same set of experiments for 1500 clients too. Given 1500 clients, FC-D and FC-TS may or may not be able to support the desired average/transient delay, as the system dynamics may deviate from the model derived via piecewise linear approximation and SYSID using 1200 clients.

Performance Results. Figures 12 and 13 show the average service delay for 1200 and 1500 clients, respectively. Notably, all the PI and fuzzy controllers support the desired *average* service delay when bursty workloads are given. FZ-D and FZ-TS further reduce the delay by carefully adjusting the backlog based on the measured performance error and change in error.

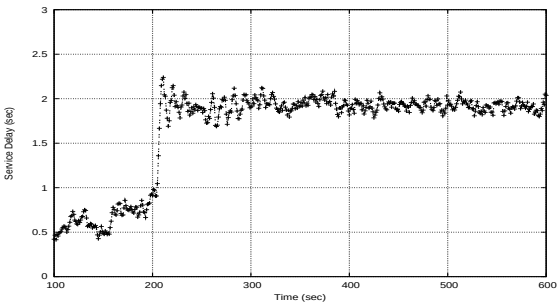
Figures 14 and 15 show the *transient* service delay for 1200 and 1500, respectively. Notably, FZ-D and FZ-TS meet the desired 2s delay bound with no delay overshoots. These results show the effectiveness of fuzzy logic control, which directly maps system behaviors to fuzzy control rules.

Although the transient delays of FC-D and FC-TS in Figure 14 do not exceed 2.5s (specified in Table 1), they often exceed 2s. FC-TS enhances the transient performance via load smoothing in addition to linear admission control; however, its delay in Figure 14 exceeds 2s several times. Due to the increased number of clients concurrently submitting service requests, FC-D and FC-TS in Figure 15 show more overshoots than they do in Figure 14.

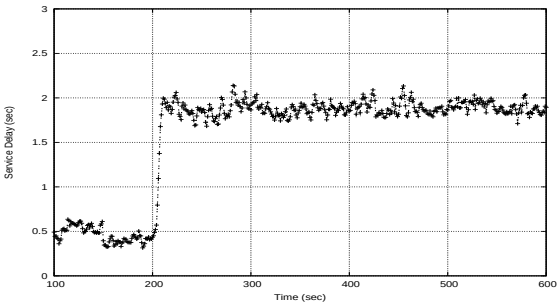
FZ-D and FC-TS converge relatively slow to the 2s set-point compared to FC-D and FC-TS in Figures 14 and 15. This is because when we design the fuzzy controller in Section 5, we aim to minimize overshoots at the cost of the relatively slow convergence rate. We trade the convergence rate for minimal overshoots, since the overshoot and settling time are two conflicting performance goals [3], [4]. If the fuzzy controller adjusts the backlog aggressively, it can achieve the quick convergence to the set-point. However, the aggressive backlog adjustment may lead to a large number of overshoots. To minimize the overshoot, we design the membership functions of fuzzy control output shown in Figure 6 in a fine-grained manner. In our fuzzy approaches, the backlog bound increases or decreases by at most 125 data items at a sampling instant. Note that this design decision for backlog adaptation is very fine-grained considering that data service requests for accessing 9,000-75,000 data arrive per second in our experiments. The experimental results in Figures 14 and 15 justify our design by showing no service delay overshoots in FZ-D and FZ-TS. Moreover, we have observed that, by avoiding performance fluctuations, FZ-D and FZ-TS improve the throughput by $5\%-16\%$ compared to FC-D and FC-TS as shown in Figure 16. A thorough investigation of a more sophisticated approach to supporting fast convergence to the desired performance without introducing excessive delay oscillations is reserved for future work.

TABLE 4
Tested Approaches

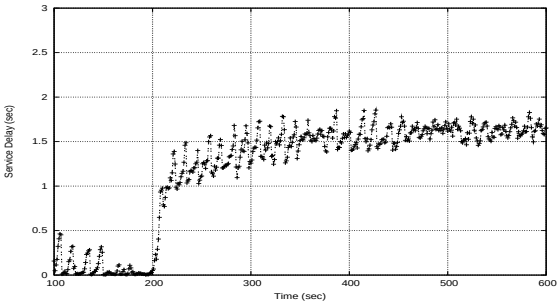
FC-D	Feedback Control(FC)−database backlog vs. delay
FC-TS	FC-D & traffic smoothing
FZ-D	Fuzzy Control(FZ)−database backlog vs. delay
FZ-TS	FZ-D & traffic smoothing



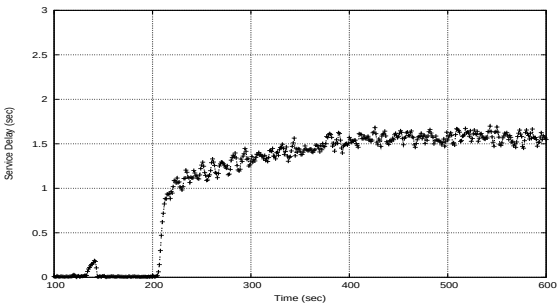
(a) FC-D



(b) FC-TS

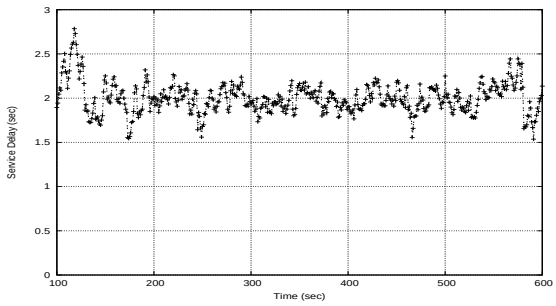


(c) FZ-D

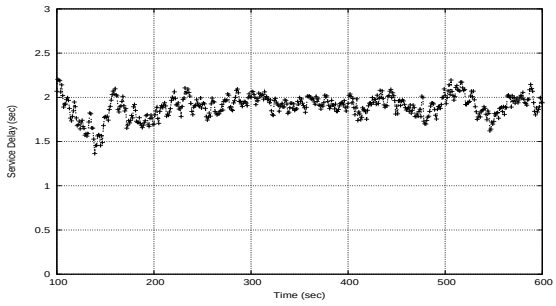


(d) FZ-TS

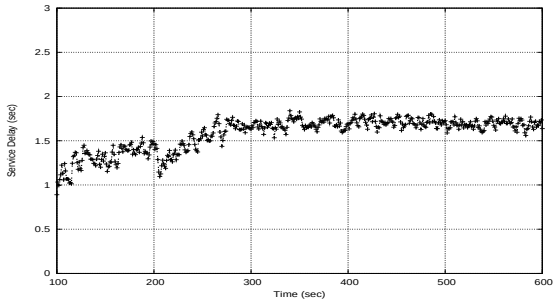
Fig. 14. Transient Service Delay (1200 clients)



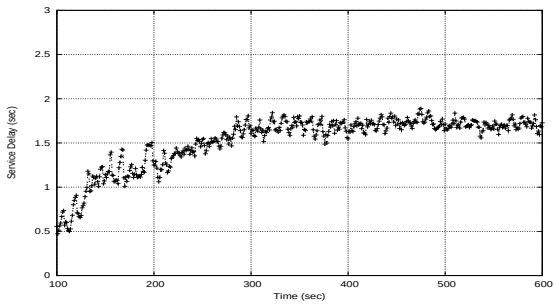
(a) FC-D



(b) FC-TS



(c) FZ-D



(d) FZ-TS

Fig. 15. Transient Service Delay (1500 clients)

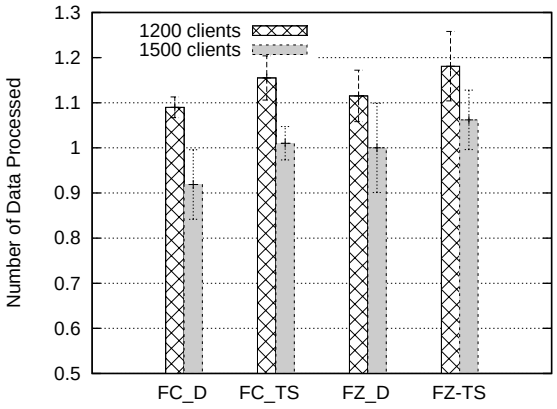


Fig. 16. Normalized average data accesses

7 Related Work

Feedback control has recently been applied to RTDB performance management; however, most of these approaches including [5], [6] are based on simulations. In general, most RTDB work—not limited to the work on feedback control of RTDB performance—is based on simulations. There have been several RTDB systems [19], [20]; however, they are either proprietary or discontinued. Thus, our work can provide valuable insights to system dynamics.

The control models used by the previous work on feedback control of RTDB performance including [5], [6] are not specific to databases but similar to [21]. Our previous work [7] applies feedback control theory to manage the performance of a real database system. In this paper, we present a new feedback control model based on the database backlog vs. delay relation, PI and fuzzy logic controllers, and a workload smoothing scheme to support the desired delay bound without degrading data freshness. The performance of our approach could be further improved, if freshness and timeliness are carefully traded off considering system behaviors and user preferences, similar to [5], [6], [22], [23].

In [24], feedback control is applied to manage the database throughput by reducing background utility work in a database, e.g., database backup or restore, under overload. Schroeder et. al. [25] determine an appropriate number of concurrent transactions to execute in a database by applying queuing and control theoretic techniques. However, no formal, detailed discussion of the control model is given. Thus, we cannot replicate their approach for comparisons. Unfortunately, none of these approaches considers to support the desired data service delay, which is critical for real-time data services.

Feedback control has been applied to real-time scheduling [21] and a web server [26]. However, they do not consider database-specific issues such as data service backlogs. In addition to feedback, feed-forward has been applied to support the desired performance in a web server [27]. Analogously, we predict the estimated database backlog before the database actually processes the submitted transactions using meta data, while adapting the burstiness of workloads, if necessary, to reduce the rejection rate before the next feedback control signal becomes available. Little prior work has been done to apply both feedback control and feed-forward techniques to real-time data services.

Fuzzy control theory [4] is originally developed to address the nonlinear characteristics, time-varying nature, and uncertainty of the system model. The use of fuzzy controllers is often justified by the non-linearity or lack of a precise mathematical model [4]. Database service delay is inherently nonlinear and stochastic. Hence, fuzzy control is an appropriate approach to managing database performance. Diao et al [28] apply fuzzy control theory to maximize the profit in an email server. eQoS [29] supports, via adaptive fuzzy control, service differentiation in a web server. However, very little prior work has applied fuzzy control to real-time performance management. In [30], [31], fuzzy control is applied to support visual tracking and achieve the specified CPU utilization in a real-time system, respectively. In this paper, we focus on improving the timeliness of data services.

8 Conclusions and Future Work

In this paper, we present several approaches for soft real-time data services:

- To support the desired data service delay, we define the notion of backlog;
- Based on the backlog model, we develop two efficient approaches for fine-grained admission control. In the first approach, we statistically identify the system model and apply linear control theory to support the desired delay bound based on the system model. In the second approach, we apply fuzzy logic control that does not rely on a mathematical model but directly tracks and controls the service delay error and change of the error. Hence, the fuzzy logic controllers supports more robust performance than the PI controllers when an approximate system model used for PI control becomes inaccurate due to workload/system setting changes; and
- To enhance the request acceptance rate under overload, we present a workload smoothing scheme. Based on the current rate of rejections, individual clients proactively reduce the workload burstiness. In this way, more client requests are accepted for small predefined smoothing delays. Notably, this approach is fully distributed requiring little intervention of the database server.

In a stock trading testbed, our approaches closely support the desired average/transient data service delay. Overall, our approaches significantly enhance the service delay and throughput compared to the tested baselines. In the future, we will continue to enhance feedback control, workload adaptation, and transaction scheduling schemes.

REFERENCES

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

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

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

[4] K. M. Passino and S. Yurkovich, *Fuzzy Control*. Addison Wesley, 1998.

[5] 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.

[6] 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.

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

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

[9] K. J. A. ström and B. Wittenmark, *Adaptive Control*. Addision Wesley, 1995.

[10] 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.

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

[12] K. D. Kang, P. H. Sin, J. Oh, and S. H. Son, "A Real-Time Database Testbed and Performance Evaluation," in *the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2007.

[13] "Transaction processing performance council," http://www.tpc.org/.

[14] "Rubis: Rice university bidding system," http://rubis.objectweb.org/.

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

[16] J. F. Kurose and K. W. Ross, *Computer Networking*. Addison Wesley, 2007.

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

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

[19] S. Kim, S. H. Son, and J. A. Stankovic, "Performance Evaluation on a Real-Time Database," in *IEEE Real-Time Technology and Applications Symposium*, 2002.

[20] Lockheed Martin, "EagleSpeed Real-Time Database Manager."

[21] 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, May 2002.

[22] H. Qu, A. Labrinidis, and D. Mossé, "UNIT: User-centric Transaction Management in Web-Database Systems," in *the 22nd International Conference on Data Engineering*, 2006.

[23] H. Qu and A. Labrinidis, "Preference-Aware Query and Update Scheduling in Web-databases," in *the 23rd International Conference on Data Engineering*, 2007.

[24] S. Parekh, K. Rose, Y. Diao, V. Chang, J. Hellerstein, S. Lightstone, and M. Huras, "Throttling Utilities in the IBM DB2 Universal Database Server," in *Proceedings of the 2004 American Control Conference*, vol. 3, June 2004, pp. 1986–1991.

[25] 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.

[26] 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, September 2006.

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

[28] Y. Diao, J. L. Hellerstein, and S. Parekh, "Using Fuzzy Control to Maximize Profits in Service Level Management," *IBM Systems Journal*, vol. 41, no. 3, 2002.

[29] J. Wei and C.-Z. Xu, "eQoS: Provisioning of Client-Perceived End-to-End QoS Guarantees in Web Servers," *IEEE Transactions on Computers*, vol. 55, Dec. 2006.

[30] B. Li and K. Nahrstedt, "A Control-Based Middleware Framework for Quality of Service Adaptations," *IEEE Journal on Selected Areas in Communications*, September 1999.

[31] M. H. Suzer and K. D. Kang, "Adaptive Fuzzy Control for Utilization Management," in *International Symposium on Object/component/service-oriented Real-time distributed Computing*, 2008.

Kyoung-Don Kang is an Assistant 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.

Yan Zhou received the Bachelor's degree in computer science from the Huazhong University of Science and Technology, China. He has received the Master's degree in computer science from the State University of New York at Binghamton in 2008 and he is currently pursuing Ph.D. in the same department. His research interests include real-time data management and QoS support in real time data services.

Jisu Oh is a PhD candidate in computer science at the State University of New York at Binghamton. She received the BS degree in computer science from the Kyoungpook National University, Korea, in 2000. Her research interests include real-time data services and wireless sensor networks. She is a student member of the IEEE.