Deadline Assignment and Feedback Control for Differentiated Real-Time Data Services

Yan Zhou and Kyoung-Don Kang

Abstract—It is challenging to process real-time data service requests, such as online trade or traffic monitoring requests, within their deadlines, while providing differentiated real-time data services. To address the problem, we present new approaches to 1) assigning deadlines to real-time data service requests based on their data access needs and service classes to differentiate real-time data services when the system is busy and 2) closely supporting the specified target delay to deadline ratio (DDR)−the ratio of actual data service delays to deadlines−via feedback control even in the presence of dynamic workloads. Further, we have actually implemented our approaches by extending an open source database unlike most existing work on real-time databases. The experimental results show that our approach closely supports the target DDR bound and service differentiation requirements among the service classes unlike the tested baselines representing the current state of the art in real-time database research.

Index Terms—Differentiated Real-Time Data Services, Deadline Assignment, Feedback Control

1 Introduction

In a number of data-intensive real-time applications, such as stock trading, traffic monitoring, or target tracking, it is critical for a real-time database (RTDB) to process data service requests within their deadlines using fresh temporal data representing the current real world status. It is also desirable for an RTDB to allow different users to subscribe to a number of different service classes, e.g., gold, silver, and bronze classes, which offer differentiated delays (and costs). However, the related work on differentiated real-time data services is relatively scarce despite the importance [2]–[5].

Data service requests in certain RTDB applications, e.g., target tracking or agile manufacturing, may have deadlines determined by the nature of the application. However, in an open system such as an online trade or traffic information system, users may simply assign arbitrarily short deadlines to their data service requests, if they are allowed to do it. As a result, the system can be severely overloaded and provide poor services. However, relatively little work has been done to investigate how to assign deadlines to real-time data service requests [1], [6]–[8]. Even less work has been done to support *differentiated deadline assignment* to real-time data service requests.

To address the problem, we estimate the size of a real-time transaction or query (i.e., read-only transaction) in terms of the number of data accesses necessary to process the transaction. Based on the transaction size estimates and service classes to which the users who submitted the transactions belong, deadlines are assigned to the transactions in a differentiated manner to meet the desired relative delay ratio, e.g., $1 : 2 : 3$ among the gold, silver, and bronze classes. After assigning deadlines to data service requests, we schedule them using the optimal earliest deadline first (EDF) scheduling algorithm that can meet all deadlines unless the system is overloaded [9]. Thus, high class users will receive favorable services in terms of relative delays when they access the similar number of data as low class clients do on average.

However, it is challenging to process real-time transactions in a timely, differentiated manner, because the workload may change dynamically when the real world status, e.g., the market or traffic status, changes. To address the challenge, we apply control theoretic techniques very effective to support the desired performance in dynamic environments [10], [11]. To design a feedback controller, one has to model the controlled system, e.g., an RTDB. In this paper, we model RTDB dynamics in terms of the relation between the *database backlog and delay to deadline ratio (DDR)* that represent the total number of data to process and the ratio of the actual data service delays to the deadlines. We take this approach, as the backlog well represents the database load. Also, the DDR generally increases as the backlog increases or vice versa. It indicates how close to the deadlines are data service requests completed in a continuous, fine-grained manner. Based on the RTDB model, we design a feedback control system to *closely support the desired DDR* even in the presence of dynamic workloads. In the feedback control loop, we systematically apply *admission control* to user data service requests, if necessary, to support the target DDR by avoiding

This work was supported, in part, by NSF grant CNS-1117352. An early version of this paper [1] was presented at the 22nd Euromicro Conference on Real-Time Systems in 2010.

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

overload conditions.

Our approach contrasts to most existing methods for differentiated real-time data services [2]–[5] that design feedback controllers based on a real-time system model [12] not derived for RTDBs. Unlike our approach, most existing work on differentiated real-time data services [2]–[5] employs a deadline miss ratio controller, a utilization controller, or both. However, a miss ratio controller saturates at 0 when the system is underutilized. On the other hand, a utilization controller saturate at 1 when the system is overloaded [13], [14]. Thus, they are subject to asymmetry and instability problems [13], [14]. Notably, our DDR metric is based on the tardiness quantile metric originally defined in [14]. However, our work is different from [14] in that we model RTDB dynamics in terms of the backlog vs. DDR relation and design a feedback control scheme based on the model. Also, we consider the problem of assigning differentiated deadlines to real-time data service requests unlike [14].

For performance evaluation, we have designed and implemented an *RTDB prototype*, called Chronos-DS, to implement our approaches to differentiated deadline assignment and DDR control by extending Berkeley DB [15]. Our work contrasts to most existing work on RTDBs based on completely theoretic analysis or high-level simulations not actually implemented in a database system [16]. The performance evaluation results show that our approach closely supports the target DDR set-point for dynamic workloads. Our approach differentiates the delay for servicing requests of different classes approximately in proportion to the specified degree of differentiation, such as 1:2:3 among three classes.

Compared to the baselines, which model the state-of-the-art approaches to real-time data management, our approach substantially increases the *timely throughput*—the number of data processed within the deadlines. Notably, the *total throughput*—the total number of data accessed by the transactions that commit within and after their deadlines—of our approach is similar to the one provided by the open-loop approach that always accepts all real-time data service requests. Thus, we do *not simply reject* a large fraction of real-time data service requests to enhance the timely throughput. Instead, we support systematic approaches to *RTDB-aware differentiated deadline assignment and DDR control*. Finally, the overhead of our approach is acceptable. According to our measurement using `/proc/stat` in Linux, our approach consumes approximately 2% CPU utilization and less than 5KB memory.

The remainder of this paper is organized as follows. An overview of our system architecture and approaches to delay estimation, deadline assignment, and DDR control is given in Section 2. The design of an RTDB prototype and transaction size estimation in the prototype as a case study are described in Section 3. The design and tuning of the feedback-based DDR control scheme is discussed in Section 4. Performance evaluation results are presented in Section 5. Related work is discussed in Section 6. Finally, Section 7 concludes the paper and discusses future work.

2 An Overview of the System

In this section, an overview of the system architecture, differentiated deadline assignment method, and DDR control scheme is given.

2.1 System Architecture

Figure 1 shows the architecture of Chronos-DS, built on top of Berkeley DB [15]. In this paper, we schedule transactions in an EDF manner. For concurrency control, we apply 2PL (two phase locking) supported by many database systems.

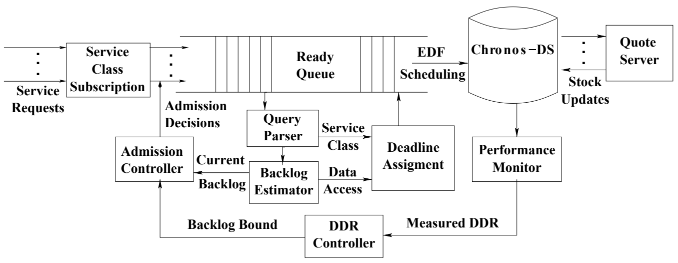


Fig. 1. System Architecture

In Chronos-DS, temporal data whose values change in time (e.g., stock prices or sensor data) are updated periodically to support the freshness of data, which is a common practice in RTDBs [16]–[18]. Each temporal data item is updated at every update period defined for the data object. To maintain the freshness of temporal data, we follow the *half-half principle* [16], [17], in which the deadline of a temporal data update is equal to its update period that is half of the validity interval for which the temporal data is considered fresh. As the workload for periodic temporal data updates is known *a priori* and fixed, we reserve dedicated threads to periodically update temporal data. In this paper, all periodic update transactions are always scheduled to support the temporal consistency of data. Admission control is only applied to user requests, if necessary, to support the desired DDR.

At each sampling point, the DDR controller in Figure 1 adapts the database backlog bound indicating the total number of data that can be processed in a timely manner, if necessary, to support the target DDR, S_t, based on the DDR error, which is the difference between S_t and the DDR measured by the performance monitor. For example, if the DDR is smaller than S_t at a sampling point, the backlog bound is increased to admit more requests in the next sampling period to avoid system underutilization or vice versa.

Upon the arrival of a data service request from a user, the backlog estimator computes the estimated number of data to be accessed by the request. The admission controller in Figure 1 admits the service request, if the estimated total backlog does not exceed the backlog bound computed by the DDR controller after accepting the new request. Otherwise, a busy signal is returned to the client, which can resubmit the request later.

If a real-time data service request is admitted, it is assigned an explicit deadline based on its data needs, i.e., the estimated number of data to access, and the service class to which the client that issued the request is subscribed as shown in Figure 1. After the deadline assignment, the request is added to the ready queue.

2.2 Execution Time Estimation and Differentiated Deadline Assignment

To derive individual deadlines for real-time data service requests, we first measure the average delay for accessing a single data item, u. To do this, we measure the average delay for a single data access offline for N (e.g., 1,000,000) data service requests:

$$u = \frac{\sum_{i=1}^{N} s_i}{\sum_{i=1}^{N} n_i} \tag{1}$$

where s_i and n_i are the processing delay and size of request i, respectively.

When the i^{th} data service request arrives, our approach estimates the number of data accesses n_i needed to process the request based on the RTDB schema. This approach is feasible, because the schema of a database is usually known at the database design time [19]. Further, transactions/queries are often predefined or canned to meet timing constraints in RTDBs [16], [17]. For example, online trades or traffic information services are usually provided through relatively short standardized transactions [16], [17], [20]–[22]. (A more detailed discussion of transaction size estimation via a case study is given in Section 3.) Based on u and n_i, we compute the estimated delay for processing the i^{th} data service request, σ_i, as follows:

$$\sigma_i = u \times n_i \tag{2}$$

To support differentiated real-time data services, we associate slack factors in proportion to the service differentiation requirements, D, with the service classes. For example, if there are three classes and the differentiation requirements $D = D_1 : D_2 : D_3$ = 1 : 2 : 3 according to the service level agreement (SLA), the corresponding slack factors F_1, F_2, and F_3 used in our approach to assign deadlines to user service requests belonging to the three classes meet the following condition: $F_1 : F_2 : F_3$ = 1 : 2 : 3. If request i from a user subscribed to class j arrives at time v_i, its deadline d_i is computed as follows:

$$d_i = v_i + F_j \times \sigma_i \tag{3}$$

Thus, a short deadline will be assigned to a request, if it belongs to a high class and accesses a small number of data. In this way, we intend to favor users belonging to a high service class, while preventing them to monopolize the system by continuously issuing large data access requests. The relative service differentiation requirement D is supported when the system is busy, if high class users access the similar number of data as low class clients do on average.

2.3 Feedback-based DDR Control

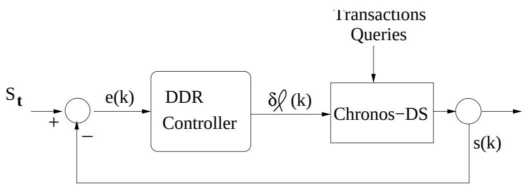


Fig. 2. DDR Control Loop

As we use EDF scheduling and the deadline of an individual request is assigned in proportion to the estimated delay and class-wise slack factor, every deadline will be met unless the workload exceeds the system capacity. Hence, data service requests will finish upon their deadlines when the system is fully utilized, meeting the differentiation requirements D. In practice, however, there could be delay estimation errors and dynamic data/resource contention, resulting in deadline misses. Therefore, in this paper, we support the desired DDR via systematic feedback control whose overall structure is shown in Figure 2.

In this paper, the *DDR* of the i^{th} data service request is formally defined as:

$$t_i = \frac{s_i}{d_i - v_i} \tag{4}$$

where s_i is the actual delay for servicing the i^{th} data service request. d_i and v_i are the deadline and the arrival time of the request, respectively. To receive a real-time data service, a client needs to establish a transmission control protocol (TCP) connection with the RTDB. The service delay s_i is the sum of the TCP connection establishment delay, queuing delay in the ready queue, and processing delay inside the database. In our approach, the DDR of an individual transaction is computed when the transaction finishes. Database systems need to handle all transaction commits/aborts [19]. The overhead of DDR monitoring is minimal compared to the computational cost for processing a transaction commit or abort.

In the closed-loop system, we control the *average* DDR to be smaller than or equal to the target DDR

S_t (≤ 1) to support timely data services. Also, we systematically manage the transient DDR via feedback control. In this paper, a DDR *overshoot* is a transient DDR that exceeds S_t when workloads vary dynamically. In control theory, an overshoot, if any, is desired to not exceed a specified threshold S_v [10], [11]. Also, an overshoot needs to be canceled within a specified time interval called the settling time T_v. The settling time is usually expressed as the number of sampling periods. Notably, in Chronos-DS, the *average/transient DDR is controlled for every class* unlike most existing work on differentiated real-time data services [2]–[5].

For feedback control, we let P indicate the sampling period (e.g., 1s) for performance measurement and feedback control. The k^{th} sampling point and the k^{th} sampling period are equal to the time kP and time interval $[(k-1)P, kP)$, respectively.[1] To closely support the desired average/transient DDR, our feedback-based DDR control scheme depicted in Figure 2 executes the following procedure at every sampling point:

- At the k^{th} sampling point, the performance monitor in Figure 1 measures $c(k)$ that represents the number of the data service requests, i.e., transactions and queries, processed in the k^{th} sampling period. It also computes the DDR $t(k) = \sum_{i=1}^{c(k)} t_i / c(k)$ and DDR error $e(k) = S_t - t(k)$.
- Based on $e(k)$, the DDR controller computes the required adjustment of the database backlog bound $\delta\ell(k)$ to support the target DDR S_t. Generally, $\delta\ell(k) > 0$ to admit more requests, if $e(k) > 0$ or vice versa.
- Using $\delta\ell(k)$ derived by the DDR controller at the k^{th} sampling point, the admission controller updates the database backlog bound to be used in the $(k+1)^{th}$ sampling period:

$$\ell(k) = \lfloor \ell(k-1) + \delta\ell(k) \rfloor \quad (5)$$

where the initial control signal $\ell(0)$ is set to a small positive integer when the system starts running.

- In the $(k+1)^{th}$ sampling period, an incoming data service request is accepted, if the total estimated backlog does not exceed $\ell(k)$ after accepting the new request.

3 A Real-Time Database Prototype

In this section, we design an RTDB prototype, because there is neither a publicly available RTDB nor a standard benchmark for RTDB research [16], [17]. Also, our approach to transaction size estimation in the prototype is described as a case study.

Although our approach is not limited to a specific data-intensive real-time application, our prototype models an online stock quote system often used to motivate RTDB research [16], [17]. It provides four types of transactions: view-stock, view-portfolio, purchase, and sale for seven tables, similar to RUBiS online auction prototype modeled after eBay [21] and the TPC-W database benchmark for e-commerce [22]. In addition, we support periodic temporal data updates for real-time data services.

To estimate the size of a user data service request, we leverage the database schema information and transaction semantics as follows:

- `View-Stock`: To process this query, Chronos-DS needs to access STOCKS and QUOTES tables that hold <stock symbol, full company name, company ID>[2]. By parsing the query, Chronos-DS finds the number of companies n_c specified in the query. Chronos-DS then computes the estimated amount of data to access for service request 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`: A client issues this query to see certain stock prices in its portfolio. For each stock item in the portfolio, Chronos-DS looks up the PORTFOLIOS table that holds <client ID, company ID, purchase price, shares> to find the company IDs used to look up the QUOTES table. Thus, the estimated amount of data accessed by this query is: $n_i = |portfolio(id)| \cdot \{r(\text{PORTFOLIOS}) + r(\text{QUOTES})\}$ where $|portfolio(id)|$ is the number of stock items in the portfolio owned by the client whose ID is id.
- `Purchase`: If a client places a purchase order for a stock item, Chronos-DS first retrieves the current stock price from the QUOTES table. If the purchased stock item was not in the portfolio before the purchase, the stock item and its purchase price are appended to the portfolio. If it is already in the portfolio, Chronos-DS updates the corresponding shares. Hence, the estimated amount of data accessed by a PURCHASE transaction is: $n_i = r(\text{QUOTES}) + (|portfolio(id)| + 1) \cdot r(\text{PORTFOLIOS})$.
- `Sale`: To process a sale transaction, Chronos-DS scans the PORTFOLIOS table to look up the stock items belonging to this client's portfolio. Using the stock IDs found in the PORTFOLIOS table, Chronos-DS searches the QUOTES table to find the corresponding stock prices. After that, Chronos-DS updates the client's portfolio in the PORTFOLIOS table to indicate the sale. Thus, the estimated amount of data accessed by a SALE transaction is: $n_i = |portfolio(id)| \cdot r(\text{PORTFOLIOS}) + n_{sell} \cdot r(\text{QUOTES}) + n_{sell} \cdot r(\text{PORTFOLIOS})$ where n_{sell} is the number of stock items to sell.

To emulate users submitting data service requests, we create client threads. A client thread submits a request to the RTDB prototype and waits for the response. After that, it waits for a certain amount of

1. In this paper, we set the sampling period P = 1s. As hundreds of (or more) transactions finish in 1s in our RTDB prototype, performance measurement for P = 1s is reliable. (All the tested approaches use the same sampling period in Section5.)

2. In fact, the schema has more attributes needed for online stock quotes and trades. In this paper, we only focus on the key attributes of the schema for the clarity of presentation.

TABLE 1
Example Service Level Agreement (SLA)

Notation	Description	Desired Value
S_t	DDR Set-point	0.9
S_v	DDR Overshoot	$1.05S_t$
T_v	Settling Time	$10s$
D	Relative Delays	1:2:3

time randomly selected in a range used to model the think time, also called the inter-request time (IRT) in this paper, between two interactive requests issued by one user. In this paper, we use a range of [1.5s, 4.5s] to model the think time of an active user, similar to [21], [22]. In our testbed, 1200 client threads in a dedicated machine submit requests to Chronos-DS running on a different machine. To model typical online queries and transactions, which are generally small and standardized [20]–[22], each request accesses 60 - 100 data items.

In our RTDB prototype, 3000 stock prices are periodically updated using the threads dedicated to temporal data updates. (In the prototype, 200 threads are reserved for temporal data updates. All the remaining threads are used to serve user requests.) The update period of each temporal data item is in a range of $[0.2s,\ 5s]$ for each stock. The schema of our stock price table follows the schema of Yahoo Finance data [20]; however, we use a larger number of temporal data and higher update frequencies than those usually provided by free online quote services.

Although we do not claim that our RTDB represents all RTDBs, it closely models an interactive quote system with typical queries and transactions. Also, our RTDB prototype can be viewed as a component of clustered databases for real-time data services, in which each RTDB is in charge of processing one data partition, similar to the scalable shared-nothing architecture [23]. However, it is largely unknown how to support efficient data partitioning and load balancing in an RTDB cluster [16], [17]. A thorough investigation is reserved for future work.

In addition, Table 1 shows an example SLA, in which the target DDR set-point $S_t = 0.9$. A DDR overshoot S_v (if any) is desired to not exceed S_t by more than 5%; that is, desirably $S_v \leq 1.05S_t$. Also, any DDR overshoot is desired to be canceled within 10s. The overshoot and settling time requirements given in Table 1 are smaller than the ones used in most existing work on differentiated real-time data services [2]–[5]. Thus, this example SLA specifies relatively stringent performance requirements.

In Chronos-DS, an SLA also specifies the *service differentiation requirements* D in terms of the relative delays among the service classes. The number of service classes are application dependent; however, usually two or three classes are considered [3], [5], [24]. Users may subscribe to different service classes considering their service needs and (monetary) costs. For example, suppose that there are three service classes and D requires the relative delay ratio $D_1 : D_2 : D_3 = 1 : 2 : 3$ among the classes when the system is busy. For timely data services considering different users' needs, an appropriate SLA can be established based on user experience surveys. For example, E*TRADE provides a 2-second execution guarantee for an interactive trade (with no service differentiation) [25], because many users usually leave if the delay is longer than a few seconds. In such a case, if the middle class receives an average 2s delay, $D_1 : D_2 : D_3 = 1 : 2 : 3$ could provide acceptable differentiated real-time data services.

4 System Modeling and Controller Design

In this section, the behavior of the controlled RTDB prototype is modeled in a control theoretic manner [10], [11]. A DDR controller is designed and tuned based on the derived model. The overall procedure consists of three major steps:

1) Model the RTDB characteristics in terms of the relation between the backlog and DDR. In fact, this is the most important step in the design of a closed-loop system based on control theory [10], [11]. If the controlled system, e.g., an RTDB, can be modeled with acceptable accuracy, well-established feedback control techniques become applicable to design a closed-loop system, since feedback control is very effective to support the desired performance even if only an approximate model is available [10], [11]. In this paper, we model the RTDB by applying the system identification (SYSID) technique widely used for *black-box modeling* [10], [11]. We apply SYSID for RTDB modeling, since SYSID only requires to model the relationship between the input to and output from the controlled system, e.g., the database backlog and DDR in the RTDB. Thus, it is useful to model complex systems such as RTDBs. In Section 4.1, the RTDB is modeled via SYSID and the accuracy of the derived model is analyzed.

2) Based on the derived model, design and tune the closed-loop system for DDR control by applying standard control theoretic techniques as described in Section 4.2.

3) Evaluate the performance of the closed-loop system (Section 5). If the desired performance is not supported, repeat the previous steps to enhance the RTDB model and controller design/tuning.

4.1 Real-Time Database Modeling via SYSID

Step 1. Identify Input/Output and Formulate a Difference Equation: Intuitively, the *DDR generally increases as the database backlog increases* or vice versa. Based on this observation, we choose the database

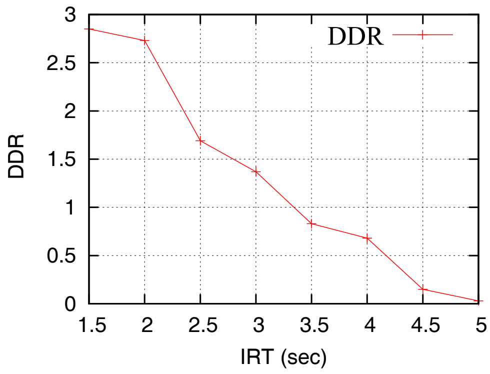


Fig. 3. Inter-Request Time (IRT) vs. DDR

backlog and DDR as the input and output of the RTDB and model the relationship between them. A common way to express the relationship between the input and output in the discrete time domain, in which most computational systems operate, is formulating a difference equation. In this paper, we model the DDR of service requests at the k^{th} sampling point via a difference equation using the DDR and backlog values measured at the previous p (≥ 1) sampling points:

$$t(k) = \sum_{i=1}^{p} \{a_i t(k-i) + b_i \ell(k-i)\} \tag{6}$$

where p is the system order [10], [11] and a_i's and b_i's are unknown model coefficients. A higher oder model is usually more accurate but more complex and subject to over-fitting to a specific workload, decreasing its adaptivity to dynamic workloads [10], [11]. Thus, the rule of thumb is to select a lowest order model that provides an enough accuracy. The main objective of RTDB modeling is deriving the model coefficients. In this section, we derive a_i's and b_i's in Eq 6 to model database dynamics by considering individual transactions, potentially accessing different numbers of data. To this end, the rest of the SYSID procedure is performed as follows.

Step 2. Select an Operating Point: To model a controlled system via SYSID, it is necessary to pick an appropriate operating point that represents the desired steady state value of the controlled variable [10], [11], such as the DDR. In this paper, we pick the DDR of 1 as the operating point, as it ideally indicates that all transactions commit upon their deadlines, while fully utilizing the system.

Step 3. Choose a Model Scope: For system modeling, we also identify a model scope to observe the DDR patterns in the vicinity of the operating point. To do this, we measure the DDR for different IRTs in our prototype described in Section 3. For SYSID, we assume that all clients belong to the highest class to model the system behavior for relatively tight deadlines. To observe the DDR for different IRTs, all requests are accepted with no admission control. Based on the profiling results, we choose the IRT range [2s, 4.5s] as the model scope, since the operating point, i.e., the DDR of 1, is observed approximately in the middle of the range as shown in Figure 3. Because the relation between the backlog and DDR in Figure 3 is near linear and the DDR does not saturate unlike the deadline miss ratio or utilization, we apply the well-established linear control theory [10] to manage the DDR.

Step 4. Collect System Dynamics Data: For SYSID, one needs to collect and analyze a large amount of system dynamics data. To this end, we run an experiment for 60 minutes, while measuring the backlog and DDR at every second. After submitting a request, a client thread waits for an IRT randomly selected within the model scope [$2s$, $4.5s$] before submitting the next request.

Step 5. Estimate Model Parameters: A primary method for deriving the model parameters in SYSID is the least square method [10], [11]. Specifically, the DDR prediction error in our model is:

$$e_p(k) = t(k) - t'(k) \tag{7}$$

where $t(k)$ is the actual DDR measured at the k^{th} sampling point and $t'(k)$ is the expected DDR predicted at the $(k-1)^{th}$ sampling point using Eq 6 with a_i's and b_i's estimated at the $(k-1)^{th}$ sampling point. In this paper, we apply the recursive least square method that iteratively estimates and adjusts the model coefficients at each sampling point to minimize the sum of the squared DDR estimation errors $\sum_{k=1}^{m} e_p(k)^2$ where m is the number of the DDR samples.

Step 6. Evaluate the Model Accuracy: To analyze the accuracy of SYSID, we use the R^2 metric [10], [11] as follows:

$$R^2 = \frac{1 - var(e_p)}{var(t)} \tag{8}$$

where $var(e_p)$ is the variance of the DDR prediction errors, $e_p(1), e_p(2), ..., e_p(m)$, and $var(t)$ is the variance of the measured DDR values, $t(1), t(2), ..., t(m)$.

The accuracy of a control model is considered acceptable, if its $R^2 \geq 0.8$ [10], [11].[3] In this paper, we choose the second order model:

$$\begin{aligned} t(k) &= 1.031t(k-1) - 0.1602t(k-2) \\ &+0.0037\ell(k-1) - 0.0029\ell(k-2) \end{aligned} \tag{9}$$

We do not consider a higher order model, because this model already provides $R^2 = 0.982$ that is near the highest possible value of 1. These results indicate the effectiveness of the backlog and DDR metrics in terms of expressing database dynamics.

Step 7. Derive the Open-Loop Transfer Function: In control theory, the relation between the input

3. If the accuracy of the derived model is not high enough, the previous steps should be redone, for example, by selecting different input/output variables and accordingly forming a new difference equation to enhance the accuracy.

and output of the controlled system is compactly expressed as a transfer function in a fractional form dividing the output by the input. Thus, the final step of our RTDB modeling is deriving a transfer function from Eq 9 to show the relation between the database backlog and DDR. To do this, we take the z-transform [10], [11] of Eq 9 to algebraically manipulate the equation in the frequency domain rather than solving partial differential equations in the time domain. A useful feature of z-transform is that z-transform of $x(k), x(k-1), x(k-2), ...$ is equal to $X(z), z^{-1}X(z), z^{-2}X(z), ...$ [10], [11]. By taking z-transform of Eq 9 and doing some algebraic manipulations using the feature, we get the following transfer function:

$$\alpha(z) = \frac{T(z)}{L(z)} = \frac{0.0037z - 0.0029}{z^2 - 1.031z + 0.1602} \quad (10)$$

where $T(z)$ is the z-transform of $t(k)$, i.e., the DDR at the k^{th} sampling point, and $L(z)$ is the z-transform of $\ell(k)$, i.e., the backlog at the k^{th} sampling point, in Eq 9.

4.2 Controller Design and Tuning

After modeling the controlled RTDB, well-established control theoretic techniques are applied to design a closed-loop system, e.g., the DDR control loop in Figure 2, to support the desired average/transient DDR as follows.

Step 1. Choose an Appropriate Controller Type: We design a *proportional and integral (PI)* controller to manage the DDR. A proportional (P) controller reacts in proportion to the control error, i.e., the difference between the DDR set-point and measured DDR in this paper. However, it cannot remove the steady state error; that is, a nonzero error is possible even when the system is in the steady state. Thus, we combine an integral (I) controller with a P controller to eliminate possible steady-state errors. A derivative (D) controller could decrease the settling time by adjusting the control input based on changes in errors. However, we do not use a D controller, because it is sensitive to noise.

At the k^{th} sampling point, the PI controller computes the control signal $\delta\ell(k)$, i.e., the database load adjustment required to support S_t as follows:

$$\delta\ell(k) = \delta\ell(k-1) + K_P[(K_I + 1)e(k) - e(k-1)] \quad (11)$$

where the error $e(k) = S_t - t(k)$ at the k^{th} sampling point. K_P and K_I are proportional and integral control gains tunable to support the desired average and transient DDR.

Step 2. Derive the Closed-Loop Transfer Function: We derive the *transfer function of a PI controller* by taking the z-transform of Eq 11:

$$\beta(z) = \frac{\Delta L(z)}{E(z)} = \frac{K_P(K_I+1)[z - 1/(K_I+1)]}{z-1} \quad (12)$$

where $\Delta L(z)$ and $E(z)$ are the z-transform of $\delta\ell(k)$ and $e(k)$, respectively.

Given the open-loop transfer function in Eq 10 and the transfer function of the PI controller in Eq 12, one can easily derive the *transfer function of the closed-loop system* using a standard method [10], [11]. Specifically, the transfer function of the closed loop system in Figure 2 is simply:

$$\gamma(z) = \frac{\alpha(z)\beta(z)}{1 + \alpha(z)\beta(z)} \quad (13)$$

Step 3. Tune the Feedback Controller: The denominator of the closed-loop transfer function is called the characteristic equation [10], [11]. Also, the roots of the characteristic equation are called closed-loop poles. To support the stability of the closed-loop system, the poles should be placed inside the unit circle [10], [11]. As the pole locations determine the overshoot and settling time too, properly placing the poles is important to tune a feedback controller.

In this paper, we tune the DDR controller by applying the Root Locus method [10]. In MATLAB, a control designer can visually locate roots inside the unit circle to support the stability of a feedback controller, while potentially moving the roots within the unit circle, if necessary, to support the desired transient performance in terms of the expected overshoot and settling time [10]. For more details about PID control and the Root Locus method, readers are referred to [10], [11].

5 Performance Evaluation

The experimental settings and average/transient performance results are discussed in this section.

5.1 Experimental Settings

For performance comparisons, we consider the following baselines:

• **Open-RTDB:** This approach extends Berkeley DB by supporting EDF scheduling. It accepts all requests, similar to state-of-the-art (proprietary) RTDBs such as [26], [27], while extending them by supporting our delay estimation and differentiated deadline assignment schemes. Thus, the main difference between Open-RTDB and our approach is that we systematically manage the DDR in the feedback loop via admission control. Notably, ad hoc admission control in proportion to the performance error, e.g., the response time error, show poor performance and fails to support the target performance when dynamic workloads are given [28]. Instead, we consider a more powerful baseline equipped with feedback-based admission control in the following.

• **PE-DIFF** (PreEmptive-DIFFerentiation): In this approach, gold and silver class requests are scheduled via EDF. However, bronze class requests are scheduled in the background and always preempted by gold or silver class queries/transactions, similar to [5]. In [5], the average and transient performance assurance is provided only to the gold class. PE-DIFF extends [5] to support the desired average and transient DDR for both the gold and silver classes. It further extends [5] by supporting our approaches to data access delay estimation, differentiated deadline assignment, and closed-loop admission control based on the backlog vs. DDR model. Hence, Open-RTDB and PE-DIFF are substantially enhanced versions of the state-of-the-art RTDB systems.

The RTDB prototype runs on a machine that has the dual core 3.8 GHz CPU and 3 GB memory. The client threads run in another machine with the 2.5 GHz CPU and 512 MB memory to submit data service requests to the RTDB. Every machine runs the 2.6.28 Linux kernel. In this paper, one experiment runs for 600s. The average performance result presented in this paper is the average of 10 runs. Each average performance data is also paired with a 95% vertical confidence interval bar.

For 80% of time, a client thread issues a query about stock prices. For the remaining 20% of time, a client requests a portfolio browsing, purchase, or sale transaction at a time. In fact, most service requests in e-commerce are queries [21], [22]. For example, RUBiS provides a browsing mix and a bidding mix, which support 85% read-only interactions and 15% requests involving writes, respectively [21]. Also, the buy-to-visit ratio in TPC-W ranges between $0.6\%-11\%$. Thus, our setting is realistic [22].

Although our approach is not limited to a specific number of service classes, data service requests are classified into three classes for the clarity of presentation, similar to [2]–[5]. For performance evaluation, we use six different workloads summarized in Table 2. For Workloads $1-4$, the inter-request time is randomly distributed in [4s, 4.5s] at the beginning of an experiment. At 200s, the range of the IRT is suddenly reduced to [1.5s, 2s] to model step workloads often used to model an abrupt workload increase in soft real-time systems dealing with dynamic workloads [28]–[30]. After 200s, the IRT stays in the new range until the end of the experiment. Therefore, at 200s, we increase the load by $2-3$ times to evaluate the adaptivity of the tested approaches to abrupt workload changes due to, for example, a sudden increase of the popularity of real-time data services. In Workloads 5 and 6, however, the IRT range of [1.5s, 2s] is used from the beginning to the end of the experiment to evaluate the performance of the tested approaches under persistent overload conditions.

For performance evaluation, we use the example SLA specified in Table 1 except for using more fine-grained $D = 1 : 1.3 : 1.6$ for Workloads $1-3$ to further stress the system. Also, we set the class-wise slack

TABLE 2
Workloads for differentiated services

Workload	D1:D2:D3	Gold, Silver, Bronze
Workload-1	1:1.3:1.6	1/3, 1/3, 1/3
Workload-2	1:1.3:1.6	2/3, 1/6, 1/6
Workload-3	1:1.3:1.6	10%, 80%, 10%
Workload-4	1:2:3	1/3, 1/3, 1/3
Workload-5	1:2:3	1/3, 1/3, 1/3 (persistent)
Workload-6	N/A	All Gold Requests

factors F_1 = 1.5, F_2 = 2, and F_3 = 2.5 in proportion to D (i.e., $F_1 : F_2 : F_3 = D_1 : D_2 : D_3 = 1 : 1.3 : 1.6$). For Workloads 4 and 5, we set $D = 1 : 2 : 3$ and accordingly set the class-wise slack factors $F_1 = 1.5, F_2 = 3$, and $F_3 = 4.5$. Finally, all requests belong to the gold class in Workload-6; therefore, a single slack factor 1.5 is used for every request.

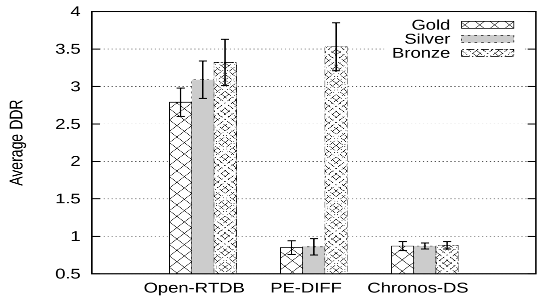
For PE-DIFF and Chronos-DS, we tune $K_P = 2.36$ and $K_I = 3.62$ via the Root Locus method to support the stability of the closed-loop system for DDR control, while closely supporting S_t, S_v, and T_v in the example SLA in Table 1 that represents relatively stringent DDR requirements.

5.2 Average Performance Analysis

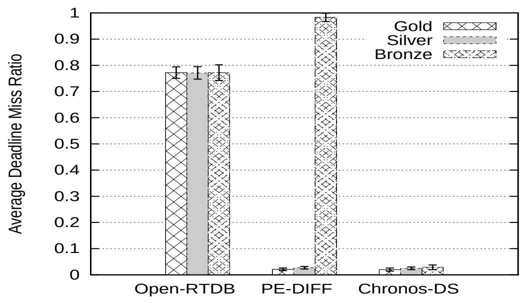
5.2.1 Average DDR, Deadline Miss Ratio, and Throughput

Figure 4 (a) shows the average **DDR** of the tested approaches for Workload-1. Open-RTDB provides the differentiated DDR across the service classes due to the differentiated deadline assignment and EDF scheduling. However, it suffers from the large DDR in every service class, because it admits all requests regardless of the current system status. In PE-DIFF, the DDR of the gold and silver classes is controlled to be below $S_t = 0.9$. Since bronze class service requests in PE-DIFF are always preempted by gold or silver class requests, the average DDR of the bronze class is larger than 3.5, which implies that a large number of deadline misses occur in the bronze class. Also, bronze class requests miss their deadlines by large margins. In contrast, Chronos-DS closely supports $S_t = 0.9$ for all the service classes as shown in Figure 4.

The **deadline miss ratio** for Workload-1 is also shown in Figure 4 (b). In Open-RTDB, the three service classes show the similarly high miss ratios, since no DDR control is applied to any service class. In Chronos-DS, the miss ratio of every service class is in the range of $2.0\% - 3.5\%$, because the DDR set-point of 0.9 is closely supported in every class. Due to the feedback-based DDR control in the gold and silver classes, the miss ratios of the gold and silver classes in PE-DIFF are only $2.0\% - 3.5\%$. However, almost all the bronze service requests miss their deadlines when the system is overloaded, since the bronze class only receives the best-effort service in the background.

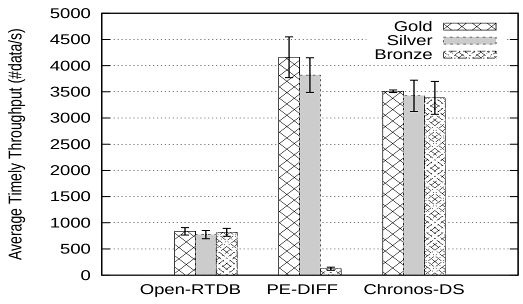


(a) DDR

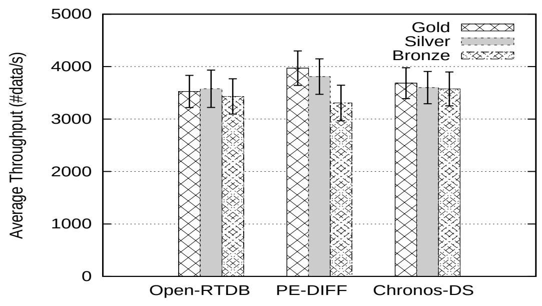


(b) Deadline Miss Ratio

Fig. 4. Average DDR and Deadline Miss Ratio for Workload-1



(a) Timely Throughput



(b) Total Throughput

Fig. 5. Average Timely and Total Throughput for Workload-1

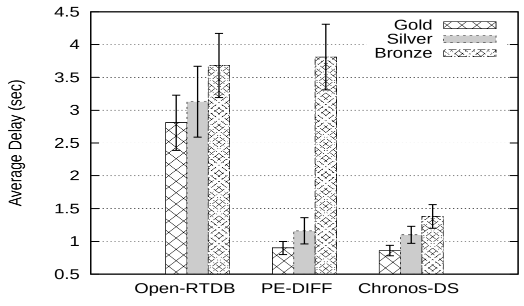
Note that this type of service differentiation may not be satisfactory. A considerable fraction of users can be dissatisfied due to substantially biased real-time data services. As a result, low class users may have to either subscribe to a higher class for additional costs or they may even unsubscribe completely.

Figure 5 (a) compares the average **timely throughput** for Workload-1. The timely throughput of the gold and silver classes of PE-DIFF is higher than that of Open-RTDB and Chronos-DS. However, the bronze class in PE-DIFF achieves the lowest timely throughput among the tested approaches, since the bronze requests in PE-DIFF are scheduled in the background as discussed before. Chronos-DS supports the largest aggregate timely throughput summed across the classes among the tested approaches. In Chronos-DS, transactions belonging to different classes are differentiated in terms of deadline assignment and EDF scheduling. However, they are treated equally in terms of feedback-based admission control to support the target DDR S_t by accepting more transactions when the DDR is smaller than S_t or vice versa regardless of their classes. Thus, it can support differentiated real-time data services without largely penalizing lower classes.

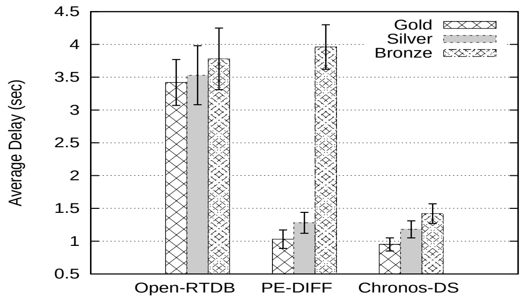
From Figure 5 (b), we observe that the average **total throughput** of Chronos-DS is similar to that of Open-RTDB. Thus, Chronos-DS significantly increases the timely throughput without sacrificing the total throughput by rejecting too many data service requests. Instead, it accepts incoming requests as long as the system capacity permits in terms of closely meeting the target DDR. PE-DIFF achieves a higher total throughput for the gold and silver classes than the other approaches do at the cost of the reduced total throughput in the bronze class. For all the six workloads, the tested approaches show similar performance in terms of the average DDR, miss ratio, timely throughput, and total throughput. Hence, the performance results with respect to these metrics for the other workloads are omitted due to space limitations.

5.2.2 Average Service Delay

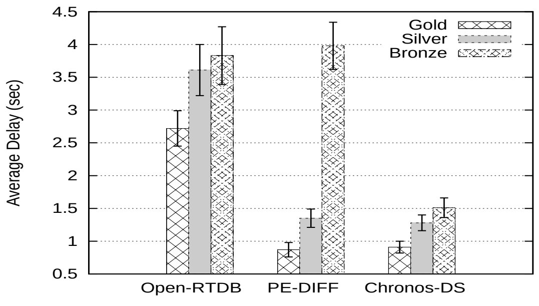
Since the class-wise **average delays** may directly affect user-perceived degree of service differentiation under heavy workloads, the average per-class delay is plotted in Figure 6 for Workloads $1-5$ that require differentiation. The average delays of the tested approaches for *Workload-1* are shown in Figure 6 (a). For Workload-1, the per-class delays of Open-RTDB are roughly 3 times larger than the delays of Chronos-DS. PE-DIFF supports significantly shorter service delays in the gold and silver classes than Open-RTDB does by controlling the DDR of service requests in these classes. However, the service requests in the bronze class suffer from excessive service delays even higher than the average delay of the bronze class in Open-RTDB, since they are always preempted by gold and silver class requests. Thus, it largely fails to support the required service differentiation ratio $D = 1 : 1.3 : 1.6$. In contrast, the average service delays of Chronos-DS are 0.86±0.08s, 1.10±0.13s, and 1.38±0.17s for gold, silver, and bronze classes, respectively. Thus,



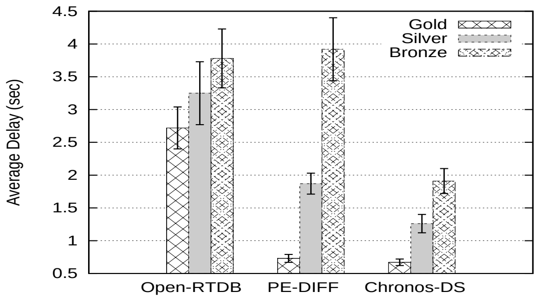
(a) Workload-1



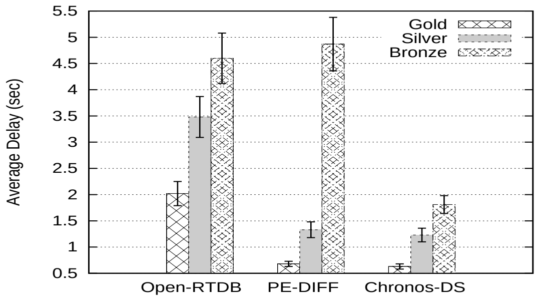
(b) Workload-2



(c) Workload-3



(d) Workload-4



(e) Workload-5

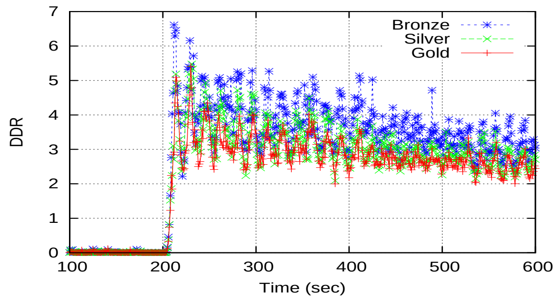
Fig. 6. Average Service Delay

Chronos-DS supports the $1 : 1.28 : 1.6$ differentiation ratio very close to D. Also, the vertical confidence interval bars of Chronos-DS are considerably smaller than the ones of the baselines. This indicates that the DDR of Chronos-DS is less variable, providing more predictable real-time data services. For Workload-3 that consists of 10% gold, 80% silver, and 10% bronze class requests (Table 2), the performance of the tested approaches is shown in Figure 6 (c). Given Workload-3, the delay of the gold class generally decreases, but the delays of the silver and bronze class slightly increase for all the tested approaches. Notably, as shown in Figure 6, only Chronos-DS can closely support D for Workloads $1 - 5$, which require service differentiation in terms of relative delays. These results experimentally verify the effectiveness of the deadline assignment and DDR control schemes of Chronos-DS for service differentiation.

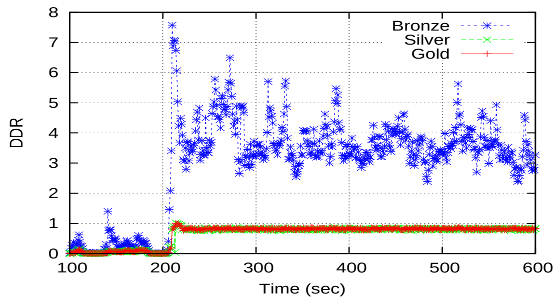
Figure 6 (d) shows the average delay provided by the tested approaches for *Workload-4*. In Chronos-DS, the delays of the gold, silver, and bronze classes are 0.67±0.05s, 1.26±0.14s, and 1.91±0.19s, respectively. Hence, the measured service differentiation ratio is $1 : 1.91 : 2.85$, which is close to the desired differentiation ratio $D = 1 : 2 : 3$ specified for this workload (Table 2), because Chronos-DS closely supports DDR set-point, $S_t = 0.9$, in every class. However, Open-RTDB and PE-DIFF largely fail to support D as shown in Figure 6 (d). For every class (especially for the silver and bronze classes), the delay of Chronos-DS is considerably shorter than the delay of the baselines as shown in Figure 6 (d).

Figure 6 (e) shows the average delay of the tested approaches for *Workload-5*. Via differentiated deadline assignment and systematic DDR control in every class, Chronos-DS achieves the differentiation ratio of $1 : 1.9 : 2.87$, closely supporting $D = 1 : 2 : 3$ despite the persistent overload conditions throughout the experiment. By comparing Figures 6 (d) and (e), we observe that the average delay in the bronze class of PE-DIFF is increased by nearly 25% for Workload 5 due to the persistent overload conditions and increased preemptions by the gold and silver classes as a result. In Figure 6 (e), the measured delay differentiation ratio of Open-RTDB ($1 : 1.75 : 2.25$) is enhanced compared to that in Figure 6 (d), because of the persistent workload with no step increase during the experiment. PE-DIFF in Figure 6 (e), however, shows the worst relative differentiation ratio among the tested approaches.

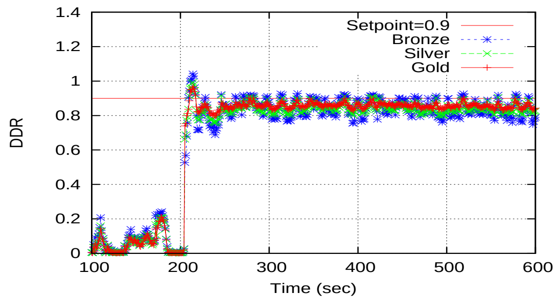
Moreover, the average DDRs and delays of the tested approaches are measured for *Workload-6* that consists of all gold class requests. Note that PE-DIFF and Chronos-DS become equivalent under this workload, because both of them intend to support the target DDR $S_t = 0.9$ for the gold class via deadline assignment, EDF scheduling, and admission control. They significantly outperform Open-RTDB in terms



(a) Open-RTDB

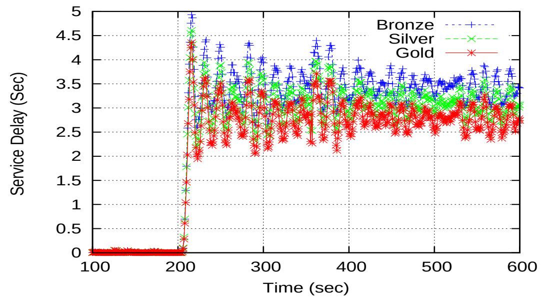


(b) PE-DIFF

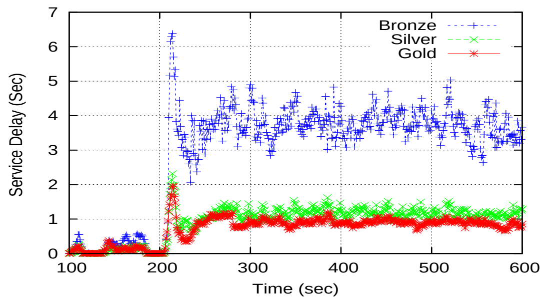


(c) Chronos-DS

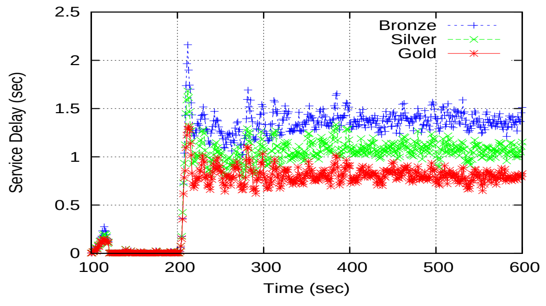
Fig. 7. Transient DDR for Workload-1



(a) Open-RTDB



(b) PE-DIFF



(c) Chronos-DS

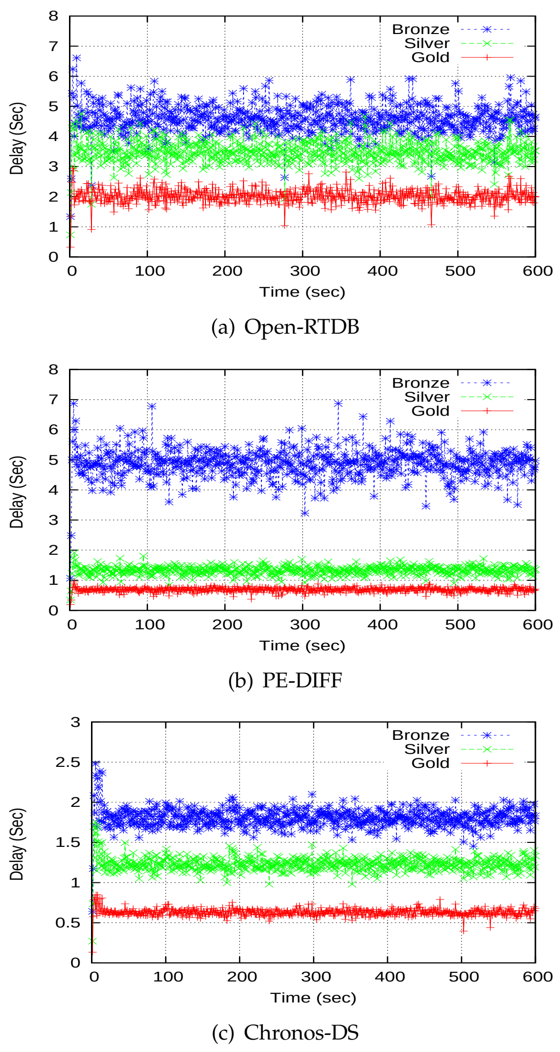
Fig. 8. Transient Delay for Workload-1

of both the service delay and DDR. The average DDR and service delay of Open-RTDB are 3.23±0.19 and 3.47±0.31s, respectively. On the other hand, the DDR and delay of PE-DIFF/Chronos-DS are 0.88±0.07 and 1.21±0.14s, respectively. Thus, our approach and PE-DIFF closely support S_t, while significantly decreasing the delay compared to Open-RTDB when all data service requests are issued by the gold class.

5.3 Transient Performance Analysis

The **transient DDR** for Workload-1 is plotted in Figure 7. In the tested approaches, the highest DDR overshoot are observed after the step workload is given at 200s. The DDR overshoots of Open-RTDB in Figure 7 (a) exceeds 6. The maximum overshoots of PE-DIFF in Figure 7 (b) are 1.07, 1.1, and 7.57, for the gold, silver, and bronze classes, respectively. It supports good transient performance for the gold and silver classes at the cost of the bronze class. As shown in Figure 7 (c), the highest DDR overshoots of Chronos-DS are 1.1, 1.12, and 1.13 for the gold, silver, and bronze classes, respectively. Also, the settling time ranges between 6s − 7s only. For the other workloads, we have observed similar transient DDR patterns for Chronos-DS; that is, Chronos-DS closely supports the desired DDR set-point, while quickly canceling overshoots. Also, PE-DIFF shows slightly higher overshoots than Chronos-DS do for the gold and silver classes, providing significantly higher overshoots for the bronze class.

Figure 8 shows the **transient service delays** of the tested approaches for Workload-1. Every service class in Open-RTDB shows much longer service delays and larger delay oscillations than every class in Chronos-DS and the gold and silver classes in PE-DIFF do. In PE-DIFF, the delay of the bronze class is substantially longer than that of the gold or silver class. Hence, PE-DIFF shows poor service differentiation in terms of transient relative delays too. As shown in Figure 8, Chronos-DS differentiates the transient delay among the service classes in a generally consistent manner, even though the transient delays vary from time to



(a) Open-RTDB

(b) PE-DIFF

(c) Chronos-DS

Fig. 9. Transient Delay for Workload-5

time. It closely supports $D = 1 : 1.3 : 1.6$ when the system becomes busy after 200s. (The transient delays for Workloads $2 - 4$ showed similar patterns. The results are not included to avoid repetitive discussions.)

In addition, Figure 9 shows the transient service delays of Open-RTDB, PE-DIFF, and Chronos-DS for Workload-5. From the figure, we observe that Open-RTDB shows the longest transient delays, while the PE-DIFF shows the worst relative delay differentiation among the tested approaches, similar to the previous results. However, Chronos-DS closely supports the desired degree of service differentiation, $D = 1 : 2 : 3$, in a consistent manner despite the persistent overload conditions in Workload-5.

In summary, Open-RTDB often fails to support the desired DDR in every service class, resulting in the considerably higher deadline miss ratios, larger delays, and lower timely throughput compared to PE-DIFF and Chronos-DS. PE-DIFF shows good performance for the gold and silver classes at the cost of the bronze class. Thus, it usually shows undesirable service differentiation in terms of relative delays. Chronos-DS closely supports the desired DDR and relative delay differentiation without penalizing lower classes in a biased manner, while achieving the higher timely throughput than Open-RTDB and PE-DIFF do. Also, it supports the similar total throughput to Open-RTDB. Hence, it does not simply reject a large number of requests to support the desired DDR. These results experimentally verify the effectiveness of Chronos-DS compared to the baselines that extend the state-of-the-art RTDBs.

6 Related Work

Although various aspects of real-time data management have been studied [16], [17], relatively little work has been done to investigate the problem of deadline assignment to user transactions in an open environment. Xiong et al. [18], [31] investigate the problem of deadline assignment to *update transactions* scheduled by the deadline monotonic and EDF scheduling algorithms to efficiently maintain the data freshness in RTDBs. They design novel approaches to assign periods and deadlines to temporal data updates to reduce the update workload. However, they do not consider the problem of assigning deadlines to user data service requests unlike our work presented in this paper. Also, feedback-based DDR control is not considered in their work. Thus, our work is complementary to theirs.

Service differentiation techniques have been studied in various computational systems such as web servers [24], [32], network routers [33], and proxy caches [34]. By providing preferred services to the high priority service class(es), limited system resources can be effectively utilized, especially under overload conditions. However, these approaches do not consider the timeliness of real-time transactions and data freshness. Thus, they are not directly applicable to support real-time data services.

Related work on differentiated real-time data services is relatively scarce [2]–[5]. Amirijoo et al. [2]–[4] present a QoS management scheme for multi-class, imprecise real-time data services. Their closed-loop approach controls the timeliness and data freshness errors within the specified per-class imprecision bounds. Kang et al. [5] provide differentiated miss ratios to the real-time data service classes via feedback control, while providing the best-effort service to the lowest class and adapting the freshness of temporal data under overload. However, most existing approaches to differentiated real-time data services [2]–[5] design feedback controllers based on a system model [12] that is not derived for RTDBs. Unlike our approach, they do not derive an RTDB system model based on RTDB semantics, such as the RTDB schema and estimated transaction sizes. They control the miss ratio or utilization subject to saturation rather than controlling the DDR, which does not suffer from the problem. They have neither considered the problem of differentiated deadline assignment nor evaluated their approaches in an actual database system.

Recently, novel approaches [6]–[8] have been developed to assign deadlines to data service requests and minimize the average tardiness without data service differentiation among service classes. However, the DDR metric is different from the traditional tardiness metric used in real-time systems. The tardiness metric indicates the difference between the finishing time and deadline, if a real-time job completes after the deadline; however, it is saturated at zero when the job completes before the deadline [9]. Thus, it can only indicate the degree of overload without capturing the degree of potential underutilization. The DDR metric is more expressive in that it can capture the degree of not only overload but also underutilization conditions. Thus, more data service requests can be admitted when the DDR is lower than the specified set-point or vice versa. Also, most these approaches aim to maximize the system revenue by processing data service requests within the deadlines, while avoiding penalties for tardy request processing. Hence, our work is complementary to them. It could be combined with these techniques to increase the revenue and reduce the late penalty by systematically controlling the DDR. A thorough investigation is reserved as a future work.

In [13], the authors apply feedback control techniques to support the target deadline miss ratio by adapting the system load. However, they observe that the miss ratio controller shows unstable performance, since the miss ratio is saturated at 0 when the system is not overloaded but abruptly increases under overload as discussed before. Other feedback-based approaches, including [35], [36], aim to indirectly control the CPU utilization that saturates at 1 to meet deadlines.

Bertini et al. [14] show that only counting the number of the missed deadlines per sampling period results in poor accuracy and large confidence intervals in terms of measuring and controlling the performance of web services. They define a novel QoS metric, called the tardiness quantile metric, and design a feedback control scheme to support the user specified tardiness quantile. Their tardiness metric is different from the traditional one [9], but similar to our DDR metric. However, these approaches do not consider the service classes and data needs of individual real-time data service requests to assign differentiated deadlines and design a closed-loop system to support the desired DDR.

A recent work, QeDB [37], presents a Multiple Input Multiple Output (MIMO) feedback control architecture to support the desired tardiness of transactions in a real-time embedded database. However, they do not consider the problem of deadline assignment based on the individual data needs of transactions in an open environment. Neither do they support differentiated real-time services. Our recent work on QoS-aware data services [28], [38] aims to support the average response time bound in a database system. However, individual transaction deadlines, DDR control, and service differentiation are not considered in [28], [38].

Data stream management systems (DSMS), including [39]–[42], support efficient processing of stream data. In DSMS, it is usually assumed that data may arrive aperiodically or even in a bursty manner and continuous queries run upon data arrivals. In contrast, RTDBs periodically update temporal data, e.g., sensor data, to support the temporal consistency of data within their confidence intervals [16]. RTDBs intend to process one-time user queries/transactions in a timely manner using temporally consistent data. Even though their data and transaction/query models are not fixed, DSMS and RTDBs are designed to deal with different scenarios [43], [44]. Thus, differentiated stream data management is beyond the scope of this paper. In addition, our feedback-based admission control scheme could be adapted and extended to support horizontal scaling to support elastic data services using more computational resources in a controlled manner under high workloads, similar to [45]. A thorough investigation of these research issues is reserved for future work.

7 Conclusions and Future Work

In a number of data-intensive soft real-time applications, it is challenging to process service requests in a timely, differentiated manner. To address the challenge, we have designed a new real-time data service model that 1) assigns differentiated deadlines to data service requests considering their service classes and estimated data access needs; and 2) provides feedback-based DDR control to support the desired target DDR even in the presence of dynamic workloads. We have designed and implemented our approach and two baselines that extend the state-of-the-art RTDBs. In the experiments, our approach closely supports the target DDR in every class, while differentiating the service delay roughly in proportion to the specified differentiation requirements. In the future, we will investigate more effective real-time data service models, feedback control methodologies, and scheduling algorithms.

References

[1] Y. Zhou and K.-D. Kang, "Deadline Assignment and Tardiness Control for Real-Time Data Services," in *Euromicro Conference on Real-Time Systems*, 2010.

[2] M. Amirijoo, J. Hansson, S. H. Son, and S. Gunnarsson, "Robust Quality Management for Differentiated Imprecise Data Services," in *the 25th IEEE Real-Time Systems Symposium*, 2004.

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

[4] M. Amirijoo, J. Hansson, and S. H. Son, "Specification and Management of QoS in Real-Time Databases Supporting Imprecise Computations," *IEEE Transactions on Computers*, vol. 55, no. 3, pp. 304–319, 2006.
[5] K. D. Kang, S. H. Son, and J. A. Stankovic, "Differentiated Real-Time Data Services for E-Commerce Applications," *Electronic Commerce Research*, vol. 3, no. 1-2, January/April 2003, combined Special Issue: Business Process Integration and E-Commerce Infrastructure.
[6] S. Guirguis, M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and K. Pruhs, "Adaptive Scheduling of Web Transactions," in *International Conference on Data Engineering*, 2009.
[7] Y. Chi, H. J. Moon, and H. Hacigümüş, "iCBS: Incremental Cost-based Scheduling Under Piecewise Linear SLAs," *Proceeding of VLDB Endowment*, vol. 4, no. 9, pp. 563–574, 2011.
[8] L. A. Moakar, P. K. Chrysanthis, C. Chung, S. Guirguis, A. Labrinidis, P. Neophytou, and K. Pruhs, "Admission control mechanisms for continuous queries in the cloud," in *International Conference on Data Engineering*, 2010.
[9] J. W. S. Liu, *Real-Time Systems*. Prentice Hall, 2000.
[10] C. L. Phillips and H. T. Nagle, *Digital Control System Analysis and Design (3rd edition)*. Prentice Hall, 1995.
[11] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. A John Wiley and Sons, Inc., Publication, 2004.
[12] 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.
[13] C. Lu, J. Stankovic, G. Tao, and S. Son, "Design and Evaluation of a Feedback Control EDF Algorithm," in *Real-Time Systems Symposium*, December 1999.
[14] L. Bertini, J. Leite, and D. Mossé, "Statistical QoS Guarantee and Energy-Efficiency in Web Server Clusters," in *the 19th Euromicro Conference on Real-Time Systems*, 2007.
[15] "Oracle Berkeley DB Product Family," available at http://www.oracle.com/database/berkeley-db/index.html.
[16] K. Ramamritham, S. H. Son, and L. C. Dipippo, "Real-Time Databases and Data Services," *Real-Time Systems Journal*, vol. 28, Nov.-Dec. 2004.
[17] K. Y. Lam and T. W. Kuo, Eds., *Real-Time Database Systems*. Kluwer Academic Publishers, 2006.
[18] 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.
[19] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd ed. McGraw-Hill, 2003.
[20] "Yahoo! Finance," http://finance.yahoo.com/.
[21] "Rubis: Rice university bidding system," http://rubis.ow2.org/.
[22] "Transaction processing performance council," http://www.tpc.org/.
[23] M. Stonebraker, "The Case for Shared Nothing Architecture," *Data Engineering Bulletin*, vol. 9, no. 1, pp. 4–9, 1986.
[24] C. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son, "A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers," in *Real-Time Technology and Applications Symposium*, 2001.
[25] "E*TRADE 2-Second Execution Guarantee," https://us.etrade.com/e/t/invest/apptemplate?gxml=exec_guara
[26] Lockheed Martin, "EagleSpeed Real-Time Database Manager."
[27] Polyhedra, "Polyhedra White Papers," http://support.polyhedra.com/embedded_database.htm.
[28] 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.
[29] C. Lu, X. Wang, and C. Gill, "Feedback Control Real-Time Scheduling in ORB Middleware," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2003, p. 37.
[30] C. Lu, X. Wang, and X. Koutsoukos, "End-to-End Utilization Control in Distributed Real-Time Systems," in *Proceedings of the 24th International Conference on Distributed Computing Systems*, 2004, pp. 456–466.
[31] M. Xiong, Q. Wang, and K. Ramamritham, "On Earliest Deadline First Scheduling for Temporal Consistency Maintenance," *Real-Time System*, vol. 40, no. 2, pp. 208 – 237, 2008.
[32] N. Bhatti and R. Friedrich, "Web Server Support for Tiered Services," *IEEE Network*, vol. 13, no. 5, pp. 64–71, September 1999.
[33] C. Dovrlois, D. Stiliadis, and P. Ramanathan, "Proportional Differentiated Services: Delay Differentiation and Packet Scheduling," in *SIGCOMM*, Aug 1999.
[34] Y. Lu, A. Saxena, and T. F. Abdelzaher, "Differentiated Caching Services; A Control-Theoretical Approach," in *the 21st International Conference on Distributed Computing Systems*, Phoenix, Arizona, April 2001.
[35] 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.
[36] 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.
[37] W. Kang, S. H. Son, and J. A. Stankovic, "Design, Implementation, and Evaluation of a QoS-Aware Real-Time Embedded Database," *IEEE Transactions on Computers*, vol. 99, no. PrePrints, 2010.
[38] K. D. Kang, J. Oh, and S. H. Son, "Chronos: Feedback Control of a Real Database System Performance," in *IEEE Real-Time Systems Symposium*, 2007.
[39] "Microsoft SQL Server 2008 R2 - StreamInsight," http://www.microsoft.com/sqlserver/2008/en/us/r2-complex-event.aspx.
[40] "Exploratory Stream Processing Systems," http://domino.research.ibm.com/comm/research_projects.nsf /pages/esps.index.html.
[41] "StreamBase," http://www.streambase.com/.
[42] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. B. Zdonik, "The Design of the Borealis Stream Processing Engine," in *CIDR*, 2005.
[43] N. Tatbul, U. Cetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker, "Load Shedding in a Data Stream Manager," in *International Conference on Very Large Data Bases*, 2003.
[44] B. Chandramouli, J. Goldstein, R. Barga, M. Riedewald, and I. Santos, "Accurate Latency Estimation in a Distributed Event Processing System," in *International Conference on Data Engineering*, 2011.
[45] H. Lim, S. Babu, and J. Chase, "Automated Control for Elastic Storage," in *International Conference on Autonomic Computing*, 2010.

Yan Zhou received his Ph.D. degree in Computer Science from the State University of New York at Binghamton in 2011. He is currently working for Rackspace, Inc. His research interests include real-time data management and QoS support for real time data services.

Kyoung-Don Kang is an Associate Professor in the Department of Computer Science at the State University of New York at Binghamton. He received a Ph.D. in Computer Science from the University of Virginia in 2003. His research areas include real-time and cyber physical systems, wireless sensor networks, and Internet of Things.