# FeedClean: Feedback-Driven Clean Utilization Management to Improve Real-Time Data Services in Dynamic Environments

Kyoung-Don Kang Department of Computer Science State University of New York at Binghamton kang@cs.binghamton.edu

# Abstract

Real-time data services can significantly increase, for example, the profit of online trades in dynamic environments such as the web by processing transactions within their deadlines. However, supporting the transaction timeliness is challenging in dynamic environments, since transactions may arrive in a bursty manner and execute longer than expected. In addition, the degree of data contention may vary. As a result, many deadlines can be missed. To address this problem, we refine real-time data service performance metrics and develop a novel feedback-based scheme to manage the clean CPU utilization, which is the difference between the aggregate utilization and wasted utilization due to data conflicts and deadline misses, in main memory realtime databases. According to the control signal computed in the feedback loop, the QoS of relatively large transactions, usually incurring more data/resource contention, can be degraded under high contention. Admission control can also be applied to incoming transactions, if necessary, to improve the real-time database performance under severe overload. By carefully managing the clean utilization, we can substantially improve the success ratio, i.e., the fraction of the submitted transactions that have been admitted and finished within their deadlines, when overloaded. In a simulation study, which can model bursty arrivals of longrunning transactions with timing constraints, our approach improves the success ratio by up to an order of magnitude compared to existing approaches.

# 1 Introduction

A real-time database (RTDB) can improve, for example, the profit, product quality, and effectiveness of ecommerce, agile manufacturing, and target tracking by processing transactions in a timely fashion. Supporting the timeliness of transactions is critical in these applications. For example, e-commerce clients are sensitive to the service delay, while a large portion of trade requests need to be processed at the back-end database servers [22]. Since existing (non-real-time) databases do not support timing constraints, they are subject to missing business opportunities or targets.

In this paper, we aim to significantly improve the performance of RTDBs operating in dynamic environments, e.g., the World Wide Web or a battle field, which often involve bursty arrivals of possibly long-running transactions that execute longer than expected with potential data conflicts [5, 16, 22].<sup>1</sup> For example, stock price updates can arrive in a bursty manner upon trades [20]. In addition, the arrival rate, execution time, and data access pattern of user transactions may vary depending on the current real world status.

Despite its importance, very little work has been done to manage the RTDB performance in such dynamic environments [3, 7, 19]. To shed light on this problem, we take a stepwise approach in which we (i) revisit and refine RTDB performance metrics; (ii) develop a novel feedbackdriven approach, called *FeedClean*, to improve the performance of a memory-resident RTDB with minimal CPU utilization wastes due to data conflicts and deadline misses;<sup>2</sup> and (iii) design a new RTDB workload model to simulate the afore-mentioned dynamic workloads and compare the performance of FeedClean to the existing baseline approaches used for performance comparisons in the literature [1, 3, 7, 8].

We compare RTDB performance management techniques using the success ratio, called *timeput* for brevity in this paper, to measure the fraction of the submitted transactions that commit within their deadlines in a sampling pe-

<sup>&</sup>lt;sup>1</sup>In this paper, the absolute size of a transaction does not determine whether or not it is long. Instead, transactions running longer than expected are considered long-running transactions, since they are known to impose more challenges on database performance management [3, 7, 16].

<sup>&</sup>lt;sup>2</sup>We consider the main memory database model such as [4, 21] that are increasingly applied to real-time data management due to the relatively high performance and decreasing memory cost. Hence, the CPU is considered the main system resource in this paper.

riod, e.g., 5 sec. (Performance metrics are formally defined in Section 2.) Note that we do not use the timeput to specify the target performance since, for example, it may not be possible for a RTDB to support a fixed target timeput 70% when the aggregate resource requirements of the transactions arriving within a sampling period is several times the system capacity in an open environment.

Unfortunately, one can not maximize the timeput by simply maximizing the utilization, since a lot of the precious utilization can be wasted due to severe data/resource contention. Transactions may execute but eventually miss their deadlines wasting the utilization. Further, transactions may have to be aborted and restarted from the beginning due to data conflicts. In the worst case, a database system may thrash by repeatedly aborting and restarting transactions without making any progress under severe data contention [14, 24].

To handle this problem, we aim to maintain a desired level, e.g., 85%, of the *clean utilization*, i.e., the difference between the current aggregate utilization and wasted utilization, to improve the timeput. We have developed a novel feedback-based approach, which is very effective to support the desired performance when the system model may include uncertainties [18], to achieve the target CPU utilization, e.g., 90%, with tolerable utilization wastes, e.g., 5%, due to contention.<sup>3</sup> In this way, FeedClean can proactively manage the RTDB performance by considering the current data/resource contention possibly incurring deadline misses and data conflicts in the future.

More specifically, the feedback control system consists of the aggregate and wasted utilization controllers that work in concert.<sup>4</sup> The utilization controller periodically monitors the current utilization and computes the control signal based on the error, i.e., the difference between the target and current utilization, in a feedback loop. When the current utilization is higher than the target, the control signal becomes negative to require the workload reduction. We assume that, in this paper, a real-time transaction consists of a mandatory part and an optional part, similar to [10, 23]. When the workload should be reduced, the QoS of a transaction currently in the system can be degraded. Only the mandatory part of a transaction is processed if its QoS is degraded. (A detailed description of the transaction model is given in Section 2.) For example, under high data/resource contention, trade transactions may only read the prices of the most important stock items (possibly pre-specified by the users), buy or sell some of those stocks if appropriate, and correspondingly update user portfolios considering the current overload and waste due to contention.

The waste controller dynamically adapts the maximum transaction size M that can be processed with the full quality of service (QoS) based on the current waste error, i.e., the difference between the desired waste threshold and the waste measured at the current sampling period. When the utilization needs to be reduced, a subset of transactions in the system whose estimated execution times are larger than M can be degraded. The motivation of this approach is that bigger transactions have a relatively high probability to abort/restart other transactions or being aborted/restarted. The waste controller further decreases Mto degrade the QoS of smaller transactions, if the waste due to data/resource contention persists over several sampling periods. Note that this approach can handle transient overloads, if any, more gracefully than existing approaches such as the transaction time-out technique [16] that simply kills transactions that run longer than a certain fixed time-out threshold under overload.

If the workload should be further reduced after potential QoS degradation, admission control is applied to incoming transactions to improve the RTDB performance by reducing the possibility of system thrashing under overload, similar to [7, 8, 14, 24]. By degrading the QoS before applying admission control, we can gracefully handle potential overloads.

Unlike most existing RTDB work (e.g., [1, 3, 7, 8]) that consider relatively smooth Poisson arrival patterns, we model bursty transaction arrivals. We also model long-running real-time transactions with different degrees of potential data contention. In the simulation study, FeedClean significantly improves the timeput compared to the baselines described in [1, 3, 7, 8], while effectively managing the clean utilization by dynamically adapting the system behavior via feedback control, QoS management, and admission control.

The rest of the paper is organized as follows. Section 2 describes the database and transaction models, defines performance metrics, and presents FeedClean. Section 3 describes the simulation techniques needed to design dynamic workloads and discusses performance evaluation results. Related work is discussed in Section 4. Finally, Section 5 concludes the paper and discusses the future work.

# 2 Performance Management in Dynamic Environments

In this section, our transaction model, performance metrics, and the architecture and behavior of FeedClean are discussed. In addition, the utilization and waste are modeled in a control theoretic manner and the feedback controllers are developed based on the models.

<sup>&</sup>lt;sup>3</sup>We do not aim to completely eliminate but aim to reduce the waste, since most databases involve transaction aborts/restarts.

<sup>&</sup>lt;sup>4</sup>We use the two controllers, because the utilization controller alone cannot control the utilization waste. Further, we have found (via simulations) that the waste controller alone may underutilize the system.

$$(D_i, QoS_i, EET_i, EU_i)$$
  
Mandatory:  
 $R_1/W_1$   
 $C_1$   
 $R_2/W_2$   
 $C_2$   
...  
Optional:  
 $R_k/W_k$   
 $C_k$   
...  

#### Figure 1. A Real-Time Transaction

# 2.1 Database and Transaction Models

We consider the main memory database model as discussed before. We consider the firm deadline semantics in which tardy transactions are aborted upon their deadline misses, for example, to avoid potential losses of profit due to market status changes. In addition, we assume that a RTDB runs a single application such as e-commerce or target tracking, since it is very hard for one RTDB to simultaneously support several data-intensive real-time applications.

As shown in Figure 1, a real-time transaction  $T_i$  is enclosed by the begin and end transaction statements.  $T_i$  is described by its relative deadline  $D_i$ , current QoS level  $QoS_i$ , estimated execution time  $EET_i$ , and estimated utilization  $EU_i$ .  $T_i$  may read or write data, i.e.,  $R_i/W_i$  in Figure 1, and do some computation, i.e.,  $C_i$  in Figure 1, based on the accessed data, if necessary, to process service requests. Further,  $T_i$  is composed of mandatory and optional parts as discussed before.

In this paper, we do not model periodic updates, since temporal data updates, e.g., write-only transactions for stock price updates, can often arrive at the database in a bursty, event-driven manner[20]. Instead, for performance evaluation, we take a general transaction model shown in Figure 1 to probabilistically vary the mix of several transaction types, including write-only, read/write, and read-only transactions, which can arrive in a bursty manner. (A detailed description of the workload model is given in Section 3.) We take this approach, because it can be more challenging to manage the RTDB performance when the arrival patterns of all types of transactions are bursty. (In the future, we will also consider more sophisticated transaction models.)

If  $QoS_i = high$  for a transaction  $T_i$ , both the mandatory and optional parts are executed; therefore,  $T_i$ 's estimated execution time  $EET_i = EET_{i_{man}} + EET_{i_{opt}}$ where  $EET_{i_{man}}$  and  $EET_{i_{opt}}$  represent the estimated execution times of the mandatory and optional parts, respectively. When  $QoS_i = low$ , only the mandatory part is executed. Thus,  $EET_i = EET_{i_{man}}$ . The estimated utilization  $EU_i = EET_i/D_i$  where  $D_i = slack \times (EET_{i_{man}} + EET_{i_{opt}})$  and slack > 0. Note that concurrently running transactions can be aborted/restarted due to read/write or write/write conflicts. Further, transaction execution times may vary depending on the accessed data values such as the current stock prices, incurring time-varying degrees of resource contention. As a result, execution time estimates may include errors.

#### 2.2 Performance Metrics

To measure the RTDB performance, we define the following performance metrics:

- The  $timeput = 100 \times \frac{N_c}{N_s}(\%)$  where  $N_c$  and  $N_s$  indicate the number of transactions *committed* within their deadlines and that *submitted* to the system in a sampling period, which is set to 5 sec in this paper to let FeedClean monitor an enough number of transactions before feedback control.
- The utilization  $U = 100 \times \frac{N_u}{N_a}(\%)$  where  $N_u$  and  $N_a$  indicate the number of the CPU cycles used for transaction processing and the number of the available cycles in a sampling period. Therefore, a higher value of U indicates a higher load, while too low a value implies an underutilization.
- The waste  $W = 100 \times \frac{N_w}{N_a}(\%)$  where  $N_w$  represents the number of the CPU cycles wasted due to deadline misses and aborts/restarts, if any, in a sampling period. Thus, a higher value of W indicates more data/resource contention. Note that  $U \ge W$ .
- The *clean utilization* C = U W(%) at a sampling instant. Thus, C represents the pure utilization used to process real-time transactions without any waste due to deadline misses or aborts/restarts.
- The  $QoS = 100 \times \frac{N_f}{N_c}$  (%) where  $N_f$  is the number of transactions that have been committed within their deadlines and processed at the full quality of service; that is, both their mandatory and optional parts are processed. In short, QoS indicates the fraction of committed transactions that received the full QoS.

For the clarity of presentation, we set the desired utilization  $U_d = 90\%$  and tolerable waste threshold  $W_d = 5\%$  in this paper. We set the desired overshoot and settling time, e.g., think time between trades, to 5% and 60 sec, respectively. Thus, an overshoot, i.e., the worst case utilization  $U_o$  or waste  $W_o$ , is such that  $U_o \leq 94.5\% (= 1.05 \times U_d)$  and  $W_o \leq 5.25\% (= 1.05 \times W_d)$  even when the system is under transient overload. Further, it is desired for a RTDB using our approach to be able to handle an overshoot, if any, in 60 sec and enter the steady state in which  $U \leq 90\%$  and  $W \leq 5\%$ .

# 2.3 FeedClean

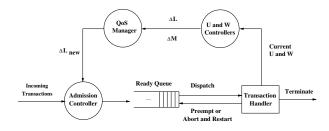


Figure 2. FeedClean Real-Time Database Architecture

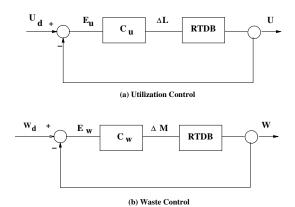
Figure 2 shows a high-level design of FeedClean. For the clarity of presentation, we also define important notations used to present FeedClean in Table 1.

Table 1. Notations for Utilization and WasteControl

Notation	Description
M	Max transaction size that can be serviced with
	the full QoS
Avg-EET	Initial value of $M$ determined offline
SP	Sampling period (= 5 sec)
$U_d$	Desired utilization
U	Current utilization
$E_u$	Utilization error
$C_u$	Utilization controller
$\Delta L$	Required workload adjustment
$\Delta L_{new}$	Required workload adjustment after possible
	QoS degradation
$W_d$	Desired (i.e., tolerable) waste
W	Current waste
$E_w$	Waste error
$C_w$	Waste controller
$\Delta M$	Required adjustment of $M$
N	Number of transactions currently in the RTDB
$QoS_i$	QoS level of a transaction $T_i$

In Figure 2, the transaction handler schedules transactions in an EDF (earliest deadline first) [11] manner. It also controls concurrency using the well-studied 2PL-HP (two phase locking high priority) protocol [1] in which a low priority transaction is aborted and restarted upon a data conflict to avoid priority inversions. A transaction can be dispatched to execute, preempted, and aborted/restarted due to data/resource contention.

The utilization and waste controllers, i.e.,  $C_u$  and  $C_w$  in Figure 3, measure the current U and W and compute the control signals, i.e.,  $\Delta L$  and  $\Delta M$  (defined in Table 1), at every sampling instant to support the desired  $U_d$  and  $W_d$ . When overloaded, i.e.,  $\Delta L < 0$ , the QoS manger degrades the QoS of the transactions that are currently executed at the high QoS level and their estimated execution times are greater than the current M. When  $\Delta L_{new} < 0$  even after possible QoS degradation, the workload is further reduced via admission control.



# Figure 3. Feedback Control of the Utilization and Waste

More specifically, FeedClean is summarized in Figure 4 and discussed as follows. At every sampling period, Feed-Clean monitors the current U and W and compare them to the desired utilization and waste, i.e.,  $U_d$  and  $W_d$ , in the feedback control loops shown in Figure 3. Based on the current utilization error  $E_u = U_d - U$ , the utilization controller  $C_u$  computes the required workload adjustment, i.e.,  $\Delta L$ , to achieve the  $U_d$  even given dynamic workloads. When overloaded, i.e.,  $\Delta L < 0$ , a transaction  $T_i$  is degraded if  $EET_i > M$ ; that is, only the mandatory part of the  $T_i$  will be executed. As a result, the CPU utilization needed to execute the optional part  $\delta U_i = EET_{i_{opt}}/D_i$  is saved and  $\Delta L$ is increased by  $\delta U_i$  as shown in Figure 4. This QoS degradation procedure is repeated until  $\Delta L$  becomes positive due to the possible degradation or there is no more transaction to degrade.

The waste controller  $C_w$  periodically monitors the current waste W to compute the control signal  $\Delta M$  based on the error  $E_w = W_d - W$ , if necessary, to adjust the maxi-

M = Avg - EETwhile (1) ł if (Clock modulo SP == 0) { /\* Compute the control signals. \*/  $E_u = U_d - U$  $\Delta L = C_u(E_u)$  $E_w = W_d - W$  $\Delta M = C_w(E_w)$  $M = M + \Delta M$ /\* QoS Degradation \*/ for  $(i = 1; \Delta L < 0 \text{ and } i \le N; i++)$ if  $(QoS_i == high \text{ and } EET_i > M)$ ł /\* Degrade the QoS of  $T_i$ . \*/  $QoS_i = low$  $EET_i = EET_i - EET_{i_{opt}}$  $EU_i = EU_i - EET_{i_{opt}}/D_i$  $\delta U_i = EET_{i_{opt}} / D_i$  $\Delta L = \Delta L + \delta U_i$ } } /\* Admission Control \*/  $\Delta L_{new} = \Delta L$ while  $(\Delta L_{new} < 0)$ { Do not admit an incoming transaction.  $\Delta L_{new} = \Delta L_{new} + EU_i$  when a transaction  $T_i$  terminates. } ł

Figure 4. A Summary of FeedClean

mum size of a transaction M that can be processed with the full QoS.  $\Delta M$  becomes negative under high data/resource contention. As a result, M is decreased to degrade more transactions in the system whose estimated execution times are greater than the new M. We assume that M is initialized as the average estimated execution time Avg-EET that can be derived offline based on the workload traces, e.g., online trade traces collected for several days.

Specifically, FeedClean handles four possible combinations of  $\Delta L$  and  $\Delta W$  as follows:

- $\Delta L \ge 0$  and  $\Delta M \ge 0$ : To avoid underutilization, admit more transactions and increase M by  $\Delta M$ .
- $\Delta L \ge 0$  and  $\Delta M < 0$ : Admit more transactions,

but decrease M by  $\Delta M$  to prepare for possible QoS degradation in the future, if necessary, to reduce the waste due to data contention. This case is possible, for example, when the CPU is underutilized, but a number of transactions in the system involve write operations incurring relatively high data contention.

- $\Delta L < 0$  and  $\Delta M \ge 0$ : Increase M by  $\Delta M$ . Degrade the QoS based on the new M. This case is possible, for example, when the CPU is highly utilized with little data conflicts possibly because most incoming transactions are read-only. Therefore, in this case, admission control rather than QoS degradation can play a more important role to reduce the workload.
- $\Delta L < 0$  and  $\Delta M < 0$ : In this case, the RTDB is overloaded and data/resource contention is high. Decrease M by  $\Delta M$ . Degrade the QoS based on the new M and apply admission control, if necessary, to further reduce the workload. Incoming transactions are not admitted until  $\Delta L_{new} \geq 0$  after a subset of currently running transactions finish.

# 2.4 Utilization and Waste Models and Feedback Controllers

To apply a control theoretic approach, we model the utilization and waste in the discrete time domain using the *z*-transform [18]. More specifically, the utilization at the  $k^{th}$  sampling instant is:

$$U(k) = U(k-1) + G_u \Delta L(k-1)$$
(1)

where  $G_u$  is the utilization gain and  $\Delta L(k-1)$  is the workload adjustment, via QoS degradation and admission control, at the  $(k-1)^{th}$  sampling period. Thus, the utilization at the  $k^{th}$  sampling period is determined by the utilization and workload adjustment at the  $(k-1)^{th}$  sampling period. In Eq 1,  $G_u$  represents the ratio between the workload adjustment performed at the previous sampling period and utilization measured at the current period. We applied various workloads to a (simulated) RTDB to linearly approximate  $G_u$  using the least square method [17]. (A detailed discussion of the workload is given in Section 3.) After taking the z-transform of Eq 1, the transfer function that describes the relation between the input to the controlled RTDB, i.e.,  $\Delta L(k-1)$ , and output from the RTDB, i.e., U(k), is:

$$T_u(z) = \frac{G_u}{z-1} \tag{2}$$

The waste at the  $k^{th}$  sampling period is:

$$W(k) = W(k-1) + G_w \Delta M(k-1) \tag{3}$$

where  $G_w$  is the waste gain and  $\Delta M(k-1)$  is the adjustment of M at the  $(k-1)^{th}$  sampling period. Hence, the waste at the  $k^{th}$  sampling period is determined by the waste and adjustment of M, which can affect QoS degradation and admission control, at the  $(k-1)^{th}$  period.  $G_w$  is derived using the least square method [17], similar to  $G_u$ . After taking the *z*-transform of Eq 3, we get the transfer function of the waste model:

$$T_w(z) = \frac{G_w}{z - 1} \tag{4}$$

Using the utilization and waste transfer functions given in Eq 2 and Eq 4, we develop the closed loop system for utilization and waste control shown in Figure 3. Given  $T_u(z)$ and  $T_w(z)$ , we can derive the closed loop transfer functions for utilization and waste control, respectively, according to control theory [18]:

$$F_{u} = \frac{T_{PI}(z)T_{u}(z)}{1 + T_{PI}(z)T_{u}(z)}$$
(5)

$$F_{w} = \frac{T_{PI}(z)T_{w}(z)}{1 + T_{PI}(z)T_{w}(z)}$$
(6)

where  $T_{PI}(z)$  is the standard transfer function of a PI (proportional and integral) controller [18]. From this, we have derived the  $F_u$  and  $F_w$  and subsequently tuned the controllers using the Root Locus method [18] to support the desired average/transient utilization and waste described in Section 2.2.

# **3** Performance Evaluation

To evaluate the performance, we have developed a RTDB simulator that models the RTDB architecture depicted in Figure 2. The admission controller, QoS manager, and feedback controllers can selectively be turned on or off for performance evaluation purposes. The main objective of the performance evaluation is to observe whether or not Feed-Clean can improve the timeput by supporting the desired clean utilization even given bursty arrivals of long-running real-time transactions with different degrees of potential data conflicts. The simulation model, baseline approaches, and performance analysis results are discussed in this section.

#### 3.1 Simulation Model

The simulation model is summarized in Table 2 and discussed in the following. The simulated RTDB has one million data items to model data-intensive real-time applications. To generate bursty transaction arrivals, each source  $S_i$  generates a sequence of transactions whose inter-arrival

Table 2. Basic Simulation Settings Parameter Value #Data Objects 1,000,000 (100,000) U(5ms, 20ms)  $EET_i$  (Estimated Exec Time)  $EET_{i_m}$  (Mandatory  $EET_i$ )  $0.5 EET_i$ (10, 20)Slack $D_i$  (Relative Deadline)  $Slack \times EET_i$ TSF (Tran. Size Factor) (1, 5)AET<sub>i</sub> (Actual Exec Time)  $TSF \times EET_i$ #Data Accesses of  $T_i$  $AET_i$  $P_w$  (Write Probability) (0.1, 0.5)

times follow the Pareto distribution, similar to the selfsimilar traffic generator [6]. For other simulation parameters such as the execution time, slack, and write probability, we have taken common values from existing real-time database work including [1, 3, 7, 8] and vary them, if necessary, to model more diverse workloads.

Each source  $S_i$  is uniformly associated with an *estimated* execution time  $EET_i$  ranging between (5ms, 20ms). The relative deadline of the transaction  $D_i = slack \times EET_i$  where the slack uniformly ranges between (10, 20). For QoS management, we assume that the estimated execution time of the mandatory part of a transaction is a half the EET needed to execute the whole transaction, i.e.,  $EET_{i_m} = 0.5EET_i$ .

To model long-running transactions that execute longer than estimated,  $S_i$  is also associated with the *actual* execution time  $AET_i = TSF \times EET_i$  where  $1 \leq TSF \leq 5$  in our experiments. Note that all the tested approaches, including FeedClean, process real-time transactions based on estimated execution times, because they may *not* have *a priori* knowledge of actual execution times (and other workload parameters) in dynamic environments.

By controlling the TSF and number of sources, we can control the load applied to the (simulated) RTDB, called *AppLoad*. More specifically, *AppLoad* = 100 ×  $\sum_{i=1}^{i=K} AET_i/D_i(\%)$  where K is the number of sources and  $AET_i$  is the actual execution time of a transaction generated by  $S_i$ . When *AppLoad* > 100\%, the workload exceeds the system capacity. Thus, the maximum possible timeput becomes less than 100%.<sup>5</sup>

The number of read/write operations in one transaction is equal to its AET. As a result, longer transactions will access more data incurring more data conflicts. In general, data and resource contention may increase as AppLoad and

<sup>&</sup>lt;sup>5</sup>If the RTDB is consistently overloaded, extra hardware resources need to be added. However, an effective RTDB performance management scheme should be able to gracefully handle unexpected transient overloads to improve the service availability. This is important, because even initially over-provisioned resources may not be enough to handle flash workloads in dynamic environments.

TSF increase. An operation of a transaction is a write operation with a tunable write probability  $P_w$ . By varying  $P_w$ , we can probabilistically vary the transaction mix as discussed before. Generally, a higher  $P_w$  is subject to more read/write and write/write conflicts. We evaluate the performance of the tested approaches by applying several  $P_w$  values ranging between 0.1 and 0.5. (Due to space limitations, we only present the results for  $P_w = 0.1$  and  $P_w = 0.5$ .)

We consider two baselines called **Admit-All** and **AC** [1, 3, 7, 8] widely used for performance comparisons in RT-DBs. Admit-All simply admits all transactions, while AC applies admission control to incoming transactions. In addition, Admit-All and AC do not degrade the QoS. For performance analysis, we set the utilization threshold of AC for admission control to 90%.

We apply three categories of workloads to Admit-All, AC, and FeedClean to compare their performance given different types of workloads: (i) nominal loads, (ii) overloads, and (iii) high data contention loads. (The performance results are discussed in Sections 3.2, 3.3, and 3.4.) For the nominal loads, we set TSF = 1. Thus, AET = EET for every transaction. We set the write probability  $P_w = 0.1$ . In addition, we increase the AppLoad from 60% to 200%. We call these worklods nominal, because the tested approach can exploit the precise execution time estimates (i.e., TSF = 1) to manage the RTDB performance, even though the AppLoad increases up to 200%.

In the overload model, we increase the TSF from 2 to 5 by 1. Accordingly, we increase the AppLoad from 200% to 500% by 100%. Note that, however, the sum of the *estimated* utilization remains as 100%. Thus, too many transactions could be admitted incurring severe data/resource contention.

To generate high data contention loads, we set  $P_w = 0.5$ and decrease the database size, similar to [1]. In this setting, the database size is reduced to  $\frac{1}{10}$  the original size. Further, we increase the estimated execution time to increase the chance of data conflicts during the transaction execution. Specifically,  $EET_i$  uniformly ranges between (5ms, 40ms). In addition, we increase TSF (AppLoad) from 2 (200%) to 5 (500%) by 1 (100%) for this set of experiments. By reducing the database size and increasing the execution time and the number of data accesses, we can increase potential data conflicts.

One simulation run lasts for 10 (simulated) minutes. Each performance data is the average of 10 simulation runs using different seed numbers. We have also derived the 90% confidence intervals. In this paper, we omit the confidence intervals because they are less than 2% in most cases. The performance evaluation results are discussed in the following subsections.

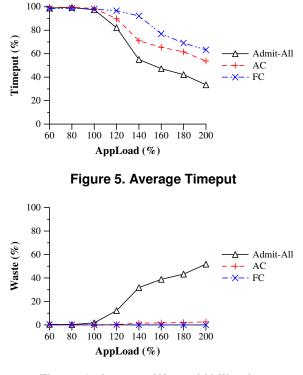


Figure 6. Average Wasted Utilization

# 3.2 Performance Results under Nominal Loads

Figures 5, 6, and 7 show the average timeput, waste, and utilization of Admit-All, AC, and FeedClean (FC) for AppLoads increasing from 60% to 200% by 20%. As shown in Figure 5, the timeput of Admit-All sharply drops as AppLoad increases, because the system is overloaded and the utilization is substantially wasted due to severe data/resource contention as shown in Figure 6. When AppLoad is 200%, the average timeput of Admit-All reduces to 33.4% and the waste increases to 51.7%. In the remainder of this paper, we drop Admit-All because of its relatively poor performance.

In contrast, AC and FC do not suffer the sharp timeput

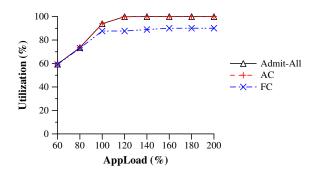


Figure 7. Average Utilization

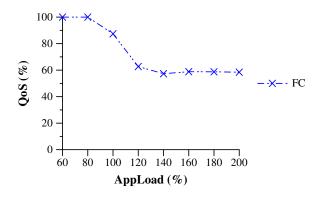


Figure 8. Average QoS

reduction or waste increase as shown in Figures 5 and 6. The timeput of AC is 53.7% when Appload = 200%. AC shows the near 0 waste until AppLoad = 120%. The waste of AC increases up to 2.6% when Appload = 200%. In this set of experiments, AC shows the good performance, since the admission control of AC is accurate when TSF = 1.

As shown in Figures 5 and 8, FC shows the highest timeput among the tested approaches at the cost of QoS degradation under overload. When AppLoad = 200%, it achieves the 63.2% timeput and 58.4% QoS meaning that approximately 58% of the committed transactions received the full QoS. In Figure 6, FC shows the near 0 waste for every tested AppLoad. As shown in Figure 7, its utilization ranges between 60% - 90%. Therefore, FC achieves (more than) the desired clean utilization (i.e., 85% as described in Section 2.2) when  $AppLoad \ge 100\%$ . In this set of experiments, FC showed no utilization or waste overshoot. Thus, we do not discuss the transient performance.

# 3.3 Performance Results under Overloads

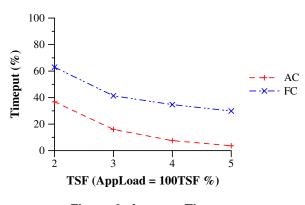


Figure 9. Average Timeput

**Average Performance.** Figures 9 and 10 show the average timeput and waste of AC and FC for increasing TSF. In Figure 9, AC, which showed the good performance in

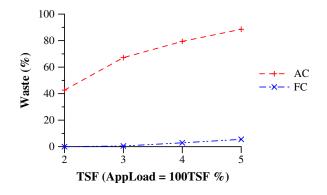


Figure 10. Average Wasted Utilization

the previous subsection, shows the poor timeput as TSF increases. When TSF = 2, its timeput is only 36.8%. Further, its timeput drops to 3.6% when TSF = 5. The timeput sharply drops, because AC does not adapt the system behavior considering the current system status. As a result, AC admits too many transactions due to high execution time estimation errors. In contrast, the timeput of FC gradually decreases from 62.9% to 29.8% as TSF increases from 2 to 5. From these results, we can observe that FC improves the timeput by up to *an order of magnitude* compared to AC (and Admit-All) when overloaded. Recall that AppLoad = 200% and 500% when TSF = 2 and 5, respectively, for this set of experiments. Thus, these timeput values also indicate that FC can gracefully handle overloads.

As shown in Figure 10, the waste of AC sharply increases reaching near 90% when TSF = 5. In contrast, FC can support the desired waste threshold, i.e.,  $W_d = 5\%$ , until TSF = 4. The waste is only 5.5% when TSF = 5. In addition, FC achieved the desired 90% utilization for all the tested TSF values. Thus, the desired clean utilization of 85% is achieved until TSF = 4 in FC. When TSF= 5, FC achieves the clean utilization of 84.5%. Hence, we observe that FC can support the desired average clean utilization even when overloaded. In contrast, AC wastes almost 90% of the utilization when TSF = 5 as shown in Figure 10.

These results show both the advantage and cost of our approach: FeedClean significantly improves the timeput and waste compared to AC (and Admit-All), while supporting the specified waste threshold when TSF  $\leq 4$ . The QoS of FC ranges between 20% - 60% for the tested TSF values. Generally, the QoS decreases as TSF increases.

**Transient Performance**. Figures 11 and 12 show the transient timeput and waste of FC when TSF = 2 and TSF = 5, respectively. (We have observed FeedClean does not suffer any utilization overshoot. Thus, we do not present the transient utilization here.)

In Figure 11, FC shows the consistent timeput mainly ranging between 50% - 70%. As shown in the figure, FC's

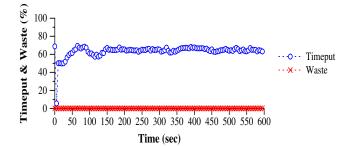


Figure 11. Transient Timeput and Waste when TSF = 2 (AppLoad = 200%)

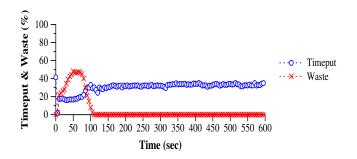


Figure 12. Transient Timeput and Waste when TSF = 5 (AppLoad = 500%)

waste is near 0 through the whole simulation. From these results, we can observe that FC supports the satisfactory transient performance when TSF = 2 by dynamically adapting the system behavior in the feedback loop according to the current utilization and waste due to data/resource conflicts. In this experiment, AC's timeput ranges between only 30% - 40%. We have also observed that AC's waste is over 40% in many cases. We do not include a graph showing AC's transient performance due to space limitations.

Figure 12 shows the transient timeput and waste of FC when TSF = 5. Initially, the timeput of FC is low, but it gradually increase and ranges between 25% - 35% within the time period of 85 sec and 600 sec. FC initially suffers the low timeput due to the flash workload that are approximately five times the system capacity. However, it quickly adapts to the workload and supports reasonable timeput after 85 sec, while achieving the near 0 waste from 110 sec. In contrast, we have observed that the transient timeput and waste of AC are near 0% and 100% through the simulation.

Table 3 shows the waste overshoot (*O*) and settling time  $(\tau)$  of FC for the tested TSF values greater than two. (The transient waste of FC is near zero when TSF = 2 as shown in Figure 11; that is, there is no waste overshoot.) The settling time, i.e., the time taken to reduce the waste below the

Table 3. Waste Overshoot and Settling Timefor Increasing TSF

TSF	3	4	5
O (%)	5.64	28	43
$\tau$ (sec)	30	85	105

specified threshold 5%, is shorter than the desired 60 sec (Section 2.2) until TSF = 3 and it is 105 sec when TSF = 5. We can observe that the every overshoot, if any, happened at the beginning of the experiments, i.e., when the flash workload was just applied, for the tested TSF values. After an initial hick-up, if any, our approach can support the desired transient performance, as shown in Figures 11 and 12, via feedback control aided by QoS degradation and admission control.

#### 3.4 Performance under Increased Data Conflicts

To show the general applicability of our approach, we have also measured the performance of FeedClean in a different setting in which the database size is only  $\frac{1}{10}$  the size used in Sections 3.2 and 3.3 to increase potential data conflicts, similar to [1]. In addition, we have increased the estimated execution times and number of data accesses as described in Section 3.1. In this set of experiments, the average utilization of FeedClean has been approximately 90%. The average timeput of FC has been decreased by 2% - 5% and the waste has been increased by 1% - 2% compared to the results described in Section 3.3. Thus, FeedClean showed the consistent average performance against the increased possibility of data conflicts.

 Table 4. Waste Overshoot and Settling Time under Increased Data Contention

TSF	3	4	5
O (%)	5.36	31	47
$\tau$ (sec)	40	85	100

When TSF = 2, there was no waste overshoot, similar to the result discussed in Section 3.3. Table 4 shows the waste overshoot and settling time for other TSF values. As shown in the table, despite the increased possibility of data conflicts, FeedClean can support the desired settling time until TSF = 3 and its settling time is 100 sec when TSF = 5, similar to the results described in Section 3.3. Overall, FeedClean significantly improves the timeput compared to the baselines, while supporting the reasonable clean utilization even given dynamic workloads, involving high errors in execution time estimates and potentially severe data conflicts.

# 4 Related Work

Vrbsky [23] applied the milestone approach [10] to realtime query processing. The accuracy of the answers monotonically increases as the computation progresses. Approximate answers to the queries can be provided, if necessary, to meet their deadlines. In the CASE-DB [15], the correctness of answers to queries can be traded off to enhance the timeliness by using the database sampling technique. Given more time, more sampling can be done to improve the correctness of the answers.

Amirijoo et al. [3] have recently applied the notion of imprecise computation in a combination with feedback control to manage the transaction timeliness and deviation of temporal data from the external environment. Kang et al. [7] propose a feedback-based RTDB QoS management scheme that can support the desired deadline miss ratio and sensor data freshness for admitted transactions. Our work is different from these work in that we focus on managing the clean utilization to improve the timeput in highly dynamic environments.

Recently, there has been a significant research interest, e.g., [9, 12, 25] to name a few, in applying feedback control to QoS management and real-time scheduling. Control theoretic approaches including [2, 13, 25] have been developed to manage the performance of web servers that may observe bursty arrivals of service requests. However, these work do not consider database issues such as data/resource conflicts and transaction aborts/restarts as we do in this paper.

In the database community, feedback control has been considered a promising approach to manage the (non-realtime) database performance [14, 24]. These work significantly improved the average database throughput via ad hoc feedback control, in which the database performance is monitored in a conceptual feedback loop and the database behavior is dynamically adapted, if necessary, to improve the average performance. However, these work do not consider transaction deadlines; therefore, they are not directly applicable to real-time transaction processing.

# 5 Conclusions

Generally, providing real-time data services in dynamic environments is essential to support important applications including e-commerce, agile manufacturing, and target tracking. However, providing real-time data services is challenging, because workloads in dynamic environments are not known *a priori*. In addition, real-time databases usually involve transaction aborts/restarts due to data/resource contention. For these reasons, non-adaptive approaches often fail. Our work presented in this paper aims to address the problem by (i) refining the performance metrics, (ii) developing a novel feedback-based approach to manage the clean utilization, (iii) designing a new RTDB workload model to simulate dynamic workloads, and (iv) performing an extensive simulation study to compare the performance of FeedClean to existing approaches. According to the performance evaluation results, FeedClean has significantly improved the timeput, while effectively managing the clean utilization. In the future, we will investigate other performance metrics and QoS management techniques to further improve real-time data services in dynamic environments. We will also investigate the QoS management issues in distributed real-time databases such as replica control and load balancing.

# References

- R. Abbott and H. Garcia-Molina. Scheduling Real-Time Transactions: A Performance Evaluation. ACM Transactions on Database System, 17:513–560, 1992.
- [2] T. Abdelzaher, Y. Lu, R. Zhang, and D. Henriksson. Practical Application of Control Theory to Web Services. In *American Control Conference*, 2004.
- [3] M. Amirijoo, J. Hansson, and S. H. Son. Algorithms for Managing QoS for Real-Time Data Services Using Imprecise Computation. In the Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA), 2003.
- [4] J. Baulier and et al. DataBlitz Storage Manager: Main Memory Database Performance for Critical Applications .
   In ACM SIGMOD - Industrial Session: Database Storage Management, 2000.
- [5] M. E. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Transactions on Networking*, 5(6), Dec 1997.
- [6] Generator of self-similar network traffic. Available at http://www.csif.cs.ucdavis.edu/kramer/code/trf\_gen2.html.
- [7] 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 En*gineering, 16(10):1200–1216, Oct. 2004.
- [8] S. Kim, S. H. Son, and J. A. Stankovic. Performance Evaluation on a Real-Time Database. In *IEEE Real-Time Tech*nology and Applications Symposium, 2002.
- [9] B. Li and K. Nahrstedt. A Control-Based Middleware Framework for Quality of Service Adaptations. *IEEE Jour*nal on Selected Areas in Communications, 17(9), 1999.
- [10] K. J. Lin, S. Natarajan, and J. W. S. Liu. Imprecise Results: Utilizing Partial Computations in Real-Time Systems. In *Real-Time System Symposium*, December 1987.
- [11] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1979.

- [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*, 23(1/2), May 2002.
- [13] 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.
- [14] A. Mönkeberg and G. Weikum. Conflict-Driven Load Control for the Avoidance of Data Contention Thrashing. In *ICDE*, 1991.
- [15] G. Ozsoyoglu, S. Guruswamy, K. Du, and W.-C. Hou. Time-Constrained Query Processing in CASE-DB. *IEEE Transactions on Knowledge and Data Engineering*, pages 865–884, Dec 1995.
- [16] N. Paranjape. Learning from (others) mistakes: Query blocking. Available at http://www.expresscomputeronline.com/20030210/techspace1.shtml.
- [17] R. Peck, C. Olsen, and J. Devore. *Introduction to Statistics and Data Analysis*. Duxbury, 2001.
- [18] C. L. Phillips and H. T. Nagle. *Digital Control System Analysis and Design (3rd edition)*. Prentice Hall, 1995.
- [19] S. H. Son and K. D. Kang. QoS Management in Webbased Real-Time Data Services. In the 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems, 2002.
- [20] The Stream Group. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin*, 26(1), 2003.
- [21] The TimesTen Team. In-Memory Data Management for Consumer Transactions The Times Ten Approach. In ACM SIGMOD, 1999.
- [22] U. Vallamsetty, K. Kant, and P. Mohapatra. Characterization of E-Commerce Traffic. *Electronic Commerce Research*, 3(1-2), 2003.
- [23] S. Vrbsky. APPROXIMATE: A Query Processor that Produces Monotonically Improving Approximate Answers. PhD thesis, University of Illinois at Urbana-Champaign, 1993.
- [24] G. Weikum, A. Moenkeberg, C. Hasse, and P. Zabback. Self-Tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering. In *VLDB Conference*, 2002.
- [25] R. Zhang, C. Lu, T. F. Abdelzaher, and J. A. Stankovic. ControlWare: A Middleware Architecture for Feedback Control of Software Performance. In *International Conference on Distributed Computing Systems*, 2002.