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, e.g., the profit of online trades by processing transactions within their deadlines. However, supporting the transaction timeliness in dynamic environments such as the WWW is challenging, since transactions may arrive in a bursty manner and execute longer than expected. As a result, the degree of data/resource contention may vary, causing deadline misses. 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 real-time databases. According to the control signal computed in the feedback loop, the QoS of relatively large transactions, which usually incur 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. In a simulation study, which models bursty arrivals of long-running 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, e.g., the profit and product quality of e-commerce and agile manufacturing. 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 [12]. Since existing (non-real-time) databases do not support timing constraints, they are subject to missing business opportunities.

In this paper, we aim to significantly improve the performance of RTDBs operating in dynamic environments, e.g., the World Wide Web, which often involve bursty arrivals of possibly long-running transactions that execute longer than expected with potential data conflicts [4, 9, 12]. For example, stock price updates can arrive in a bursty manner upon trades [11]. 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, 6, 10]. Most existing RTDB research such as [1, 3, 6, 7] usually consider workloads consisted of periodic updates and relatively smooth user transaction arrival patterns such as Poisson arrival patterns. 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 feedback-driven approach, called $FeedClean$, to improve the performance of a memory-resident RTDB with minimal CPU utilization wastes due to data conflicts and deadline misses; 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, 6, 7].

We measure RTDB performance in terms of success ratio, i.e., the fraction of the submitted transactions that commit within their deadlines in a sampling period, e.g., 5 sec. (For brevity, success ratio is called *timeput* in this paper.) 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 resource/data contention. A transaction execution can be delayed due to resource contention. Its deadline can eventually be missed wasting the utilization used to execute the transaction before the deadline miss. In addition, 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 [13].

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 to achieve the target CPU utilization, e.g., 90%, with tolerable utilization wastes, e.g., 5%, due to contention.[1] The feedback control system consists of the aggregate and wasted utilization controllers that work in concert.[2] 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. When the workload should be reduced, the QoS of a transaction currently in the system can be degraded.

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. We take this approach because bigger transactions have a relatively high probability to abort/restart other transactions or being aborted/restarted. Note that this approach can handle transient overloads, if any, more gracefully than existing approaches such as the transaction time-out technique [9] 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 [6, 7, 13]. By degrading the QoS before applying admission control, we can gracefully handle potential overloads.

Unlike most existing RTDB work (e.g., [1, 3, 6, 7]) that consider periodic updates and relatively smooth Poisson arrival patterns, we model bursty transaction arrivals that can model both temporal data, e.g., stock price, updates and user 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, 6, 7], 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.

[1] We do not aim to completely eliminate but aim to reduce the waste, since most databases involve transaction aborts/restarts.

[2] 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 under utilize the system.

2 Performance Management in Dynamic Environments

In this section, our transaction model, performance metrics, and the architecture and behavior of FeedClean are discussed. (We also model the utilization and waste in a control theoretic manner. Based on the models, we develop the feedback controllers. However, the modeling and controller development are not included in this paper due to space limitations. We refer interested readers to [5] for the related discussion.)

Database and Transaction Models. We consider the main memory database model in which transactions have firm deadlines. Tardy transactions are aborted upon their deadline misses, .e.g., to avoid potential losses of profit due to market status changes. A real-time transaction T_i is described by its relative deadline D_i, current QoS level QoS_i, and estimated execution time EET_i. T_i may read or write data and do some computation based on the accessed data. T_i is composed of mandatory and optional parts, similar to [8]. If the QoS_i of T_i is high, 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 $slack > 0$. Transactions can be aborted/restarted due to data contention and execution times may vary depending on the accessed data values, e.g., the current stock prices. As a result, execution time estimates may include errors.

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. The *clean utilization* $C = U - W$ where U is the measured CPU utilization and W is the utilization wasted due to deadline misses and aborts/restarts in a sampling period. 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, executing both the mandatory and optional parts. 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, are 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 $W \leq 5\%$.

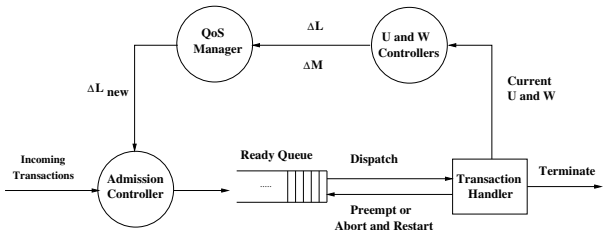

Figure 1. FeedClean Real-Time Database Architecture

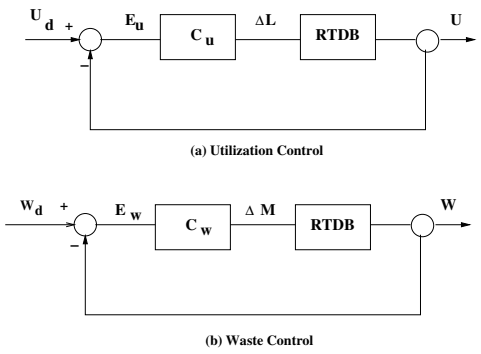

Figure 2. Feedback Control of the Utilization and Waste

FeedClean. Figure 1 shows a high-level design of FeedClean. The transaction handler schedules transactions in an EDF (earliest deadline first) 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. At every sampling period, FeedClean monitors the current U and W and compare them to the desired U_d and W_d as shown in Figure 2. Based on the current utilization error $E_u = U_d - U$, the utilization controller C_u computes the required workload adjustment, i.e., ΔL, to achieve the U_d. 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 utilization needed to execute the optional part $\delta U_i = EET_{i_{opt}}/D_i$ is saved and ΔL is increased by δU_i. The QoS degradation is repeated until ΔL becomes positive or there is no more transaction to degrade. The waste controller C_w periodically monitors the current waste W to compute the control signal ΔM based on the error $E_w = W_d - W$, if necessary, to adjust the maximum size M of a transaction that can be processed with the full QoS. ΔM 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 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 ΔL and ΔW as follows. (i) $\Delta L \geq 0$ and $\Delta M \geq 0$: To avoid underutilization, admit more transactions and increase M by ΔM. (ii) $\Delta L \geq 0$ and $\Delta M < 0$: Admit more transactions, but decrease M by ΔM 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. (iii) $\Delta L < 0$ and $\Delta M \geq 0$: Increase M by ΔM. 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. (iv) $\Delta L < 0$ and $\Delta M < 0$: In this case, the RTDB is overloaded and data/resource contention is high. Decrease M by ΔM. 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.

3 Performance Evaluation

To evaluate the performance, we have developed a RTDB simulator that models the RTDB architecture depicted in Figure 1. 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 FeedClean 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.

Simulation Model. 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 times follow the Pareto distribution, similar to [4]. 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, 6, 7] 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, $AppLoad = 100 \times \sum_{i=1}^{i=K} AET_i/D_i(\%)$ 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%.

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 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.

We consider two baselines called **Admit-All** and **AC** [1, 3, 6, 7] widely used for performance comparisons in RTDBs. 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**. 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 workloads 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. Due to space limitations, we only present the performance results under the overload conditions in this paper. We refer readers to [5] for the other performance results.

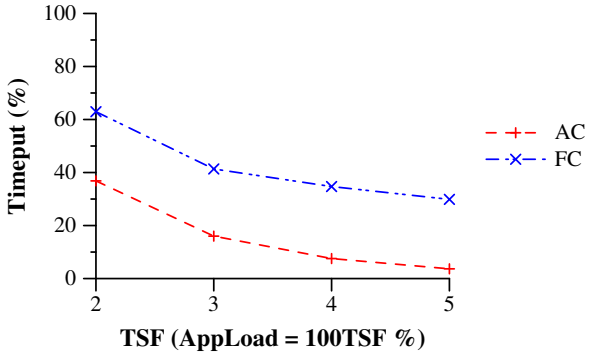

Figure 3. Average Timeput

Average Performance Under Overloads. Figures 3

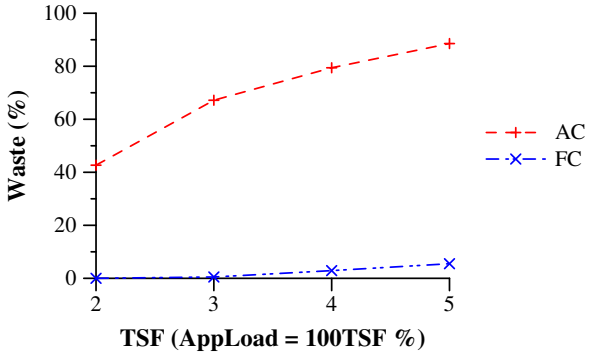

Figure 4. Average Wasted Utilization

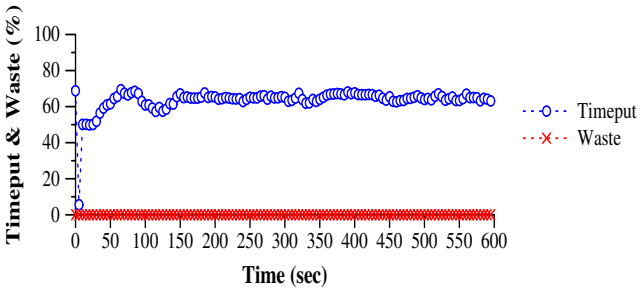

Figure 5. Transient Timeput and Waste when TSF = 2 (AppLoad = 200%)

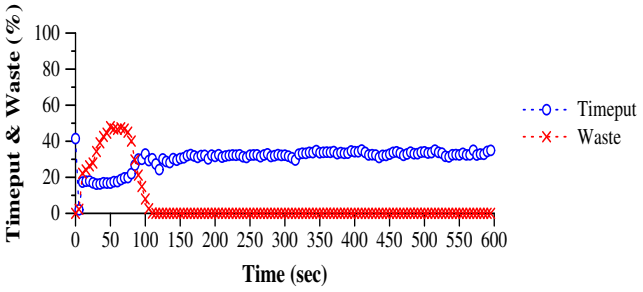

Figure 6. Transient Timeput and Waste when TSF = 5 (AppLoad = 500%)

and 4 show the average timeput and waste of AC and FC for increasing TSF. (We do not discuss Admit-All here, because its performance is relatively poor compared to AC and FC.) In Figure 3, AC 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. Thus, these timeput values also indicate that FC can gracefully handle overloads.

As shown in Figure 4, 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%. From these results, observe that FC supports the desired average clean utilization even when overloaded. In contrast, AC wastes almost 90% of the utilization when TSF = 5 as shown in Figure 4.

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 Under Overloads. Figures 5 and 6 show the transient timeput and waste of FC when TSF = 2 and TSF = 5, respectively. In Figure 5, FC shows the consistent timeput mainly ranging between $50\% - 70\%$. As shown in the figure, FC's 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 6 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 1 shows the waste overshoot (O) and settling time (τ) 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 5; that is, there is no waste overshoot.) The settling time, i.e., the time taken to reduce the waste below the specified threshold 5%, is shorter than the desired 60

Table 1. Waste Overshoot and Settling Time for Increasing TSF

TSF	3	4	5
O (%)	5.64	28	43
τ (sec)	30	85	105

sec (Section 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 5 and 6, via feedback control aided by QoS degradation and admission control.

4 Related Work

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 Ab. [6] 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. Control theoretic approaches [2] 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.

5 Conclusions

Generally, providing real-time data services in dynamic environments is essential to support important applications including e-commerce and agile manufacturing. 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

[1] 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 andApplications (RTCSA)*, 2003.

[4] 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.

[5] K. D. Kang. FeedClean: Feedback-Driven Clean Utilization Management to Improve Real-Time Data Services in Dynamic Environments. Technical Report CS-TR-05-KD01, Department of Computer Science, State University of New York at Binghamton, 2005. Available at http://www.cs.binghamton.edu/∼kang.

[6] K. D. Kang, S. H. Son, and J. A. Stankovic. Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases. *IEEE Transactions on Knowledge and Data Engineering*, 16(10):1200–1216, Oct. 2004.

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

[8] 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.

[9] N. Paranjape. Learning from (others) mistakes: Query blocking. Available at http://www.expresscomputeronline.com/20030210/techspace1.shtml.

[10] S. H. Son and K. D. Kang. QoS Management in Web-based Real-Time Data Services. In *the 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems*, 2002.

[11] The Stream Group. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin*, 26(1), 2003.

[12] U. Vallamsetty, K. Kant, and P. Mohapatra. Characterization of E-Commerce Traffic. *Electronic Commerce Research*, 3(1-2), 2003.

[13] 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.