Control-Based Real-Time Metadata Matching for Information Dissemination

Ming Chen, Xiaorui Wang, Raghul Gunasekaran, Hairong Qi
Department of Electrical Engineering and Computer Science
University of Tennessee
Knoxville, TN 37996
{*mchen11, xwang, raghul, hqi*}*@utk.edu*

Mallikarjun Shankar
Computational Sciences and Engineering
Oak Ridge National Laboratory
Oak Ridge, TN 37830
shankarm@ornl.gov

Abstract

Real-time information dissemination is of increasing importance to our society. Existing work mainly focuses on delivering information from sources to sinks in a timely manner based on established subscriptions, with the assumption that those subscriptions are persistent. However, the bottleneck of many real-time information dissemination systems is actually the matching process to continuously reevaluate such subscriptions between numerous sources and numerous sinks, in response to dynamically varying information attributes at runtime. In this paper, we propose a feedback controller to adaptively meet the response time constraints on metadata matching in an example information dissemination system. Our controller features a rigorous design based on well-established feedback control theory for guaranteed control accuracy and system stability. Empirical results on a physical test-bed demonstrate that our controller outperforms both an open-loop solution and a typical heuristic solution, by having more accurate control and better system quality of service.

1 Introduction

In recent years, real-time information dissemination has become increasingly important. For example, alert messages from stock tickers should be disseminated in a timely manner to subscribing customers. Similarly, buyers and sellers need to be matched based on their interests in e-commerce systems, and notified immediately when new business opportunities are identified. Likewise, in surveillance applications, threats detected by various sensors must be reported to appropriate authorities within certain time constraints. In these applications, data flows from numerous (*e.g.*, thousands of) sources to numerous sinks have to be channeled flexibly, efficiently and more importantly, in a real-time manner. This requirement has been generally described as *Valuable Information at the Right Time (VIRT)* [1], which emphasizes that consumers of information should receive the information that is of interest to them as soon as it is available or whenever it is requested.

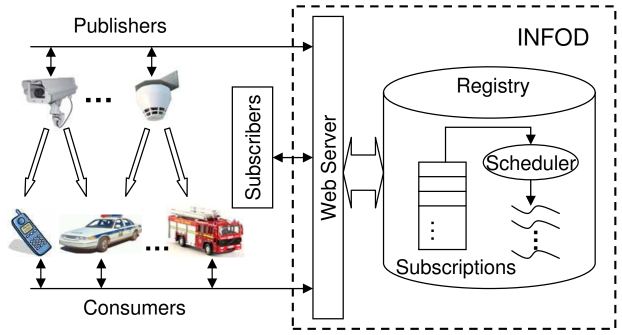


Figure 1. INFOD: an example information dissemination system

An example information dissemination system is INFOD (INFOrmation Dissemination) [2], which is a project that aims to support timely delivery of valuable information for a wide range of applications. Similar to other information dissemination systems based on the publish-subscribe paradigm, *publishers* are defined as the information sources and *consumers* are the information sinks in INFOD. As shown in Figure 1, INFOD serves as a broker to allow publishers and consumers to advertise their *attributes* and *constraints* in the system. For example, a consumer may have its location as an attribute and have a constraint on desired publishers: they must be located in Knoxville. The attributes and constraints are generally referred to as *metadata*, which is stored in a database called *registry*. *Subscribers* submit *subscriptions* on behalf of the consumers. Subscriptions have different priorities. An example of high-priority subscription is: all sensors (publishers) send their temperature data to all firefighters (consumers) within 10 miles every 2 seconds. Based on the subscription, INFOD establishes connections between publishers and consumers by finding matches between their attributes and constraints. Based on the matching results, publishers are informed where to send filtered information to the matched consumers without going through INFOD.

A common problem faced by information dissemination systems running in dynamic environments is that the registered attributes and constraints may vary at runtime. For example, a firefighter may constantly update its location attribute. As a result, all subscriptions in the registry need to be continuously reevaluated by rerunning metadata matching to ensure that the firefighter receives information from the sensors in the right locations. However, given the large number of publishers and consumers, reevaluating all subscriptions in the registry may cause severe system overload and unacceptably long delays. Therefore, it is a common practice to guarantee that the average response time of reevaluating all high-priority subscriptions is within a real-time constraint, which is referred to as *real-time metadata matching*. In the meantime, the maximal possible number of low-priority subscriptions should also be reevaluated. The number of reevaluated low-priority subscriptions is defined as the *quality of service (QoS)* of the system.

Real-time metadata matching faces two major challenges. First, execution time of reevaluating a subscription may vary significantly as the numbers of involved publishers, consumers and constraints are different for each subscription and are dynamically changing at runtime. Second, metadata updates may come either periodically or aperiodically with unpredictable time intervals. These uncertainties may lead to unpredictable delays for real-time metadata matching.

In this paper, we propose a novel feedback controller to adaptively control the *average* response time of metadata matching by manipulating the number of low-priority subscriptions to be reevaluated. The manipulated variable is called *job budget*, which includes all the high-priority subscriptions and the low-priority subscriptions selected by the controller. Specifically, the contributions of our work are four-fold:

1. We analytically model the relationship between the average response time and the job budget, and validate the model with white noise inputs.

2. We design and analyze the controller based on well-established feedback control theory for theoretical guarantees on control accuracy and system stability.

3. We prove, through both theoretical analysis and experimentation, that the desired control performance can be achieved even in the presence of runtime variations that cause the system to behave differently from the nominal system model used to design the controller.

4. We present empirical results on a physical test-bed to demonstrate that our controller outperforms both an open-loop solution and a typical heuristic solution.

The rest of the paper is organized as follows. Section 2 introduces the overall architecture of the feedback control loop. Section 3 presents system modeling and controller design and analysis. Section 4 analyzes the performance of the controller when the system model varies at runtime. Section 5 describes the implementation details of each component in the control loop. Section 6 presents the results of our experiments. Section 7 discusses related work while Section 8 concludes the paper.

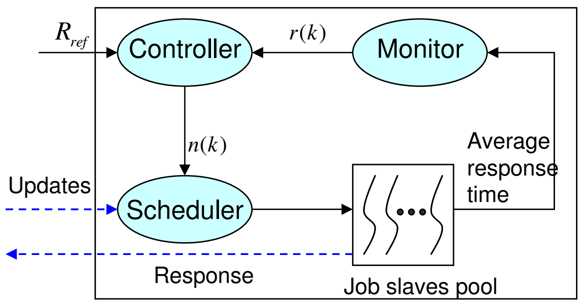


Figure 2. Feedback control architecture

2 Feedback Control Loop

In this section, we give a high-level description of our feedback control loop that controls the average response time of metadata matching by dynamically adjusting the job budget. Our controller features a rigorous design based on feedback control theory for guaranteed control accuracy and system stability [3]. The benefit of having control theory as a theoretical foundation is that we can have (1) standard approaches to choosing the right control parameters so that exhaustive iterations of tuning and testing are avoided; (2) theoretically guaranteed control performance such as accuracy, stability, short settling time, and small overshoot; and (3) quantitative control analysis when the system is suffering unpredictable workload variations. This rigorous design methodology is in sharp contrast to heuristic-based adaptive solutions that rely on extensive empirical evaluation and manual tuning [4].

As shown in Figure 2, the key components in the control loop include a centralized *controller*, an average response time *monitor* and an application-level *scheduler*. The feedback control loop is invoked periodically at the end of every control period as follows:

1. The monitor measures the average response time of *all* metadata matching processes executed in the last control period and sends the value to the controller. The average response time is the *controlled variable* of the control loop.

2. The controller calculates the appropriate job budget in the next control period based on the difference between the set point and the measured average response time. The job budget is the *manipulated variable* of the control loop.

3. Based on the calculated job budget, the scheduler schedules all the high-priority and the desired number of low-priority subscriptions to be reevaluated in the registry.

3 Controller Design

The core of our feedback control loop is the controller. In this section, we first formulate the control problem. We

Table 1. Comparison of different models

Model	n_a	n_b	R^2(estimation)	R^2(validation)
1	0	1	0.8416	0.70314
2	0	2	0.85653	0.70346
3	1	1	0.8529	0.7034
4	1	2	0.85726	0.70364

then establish a mathematical model for the open-loop system. Finally, we design our controller based on the system model.

3.1 Problem Formulation

We first introduce the following notations.

- T: Control period of the feedback control loop. The period is selected in such a way that each control period can include multiple metadata updates.
- R_{ref}: Set point for average response time. The set point is selected to be slightly shorter than the desired time constraint to have some leeway because we are controlling the average response time.
- $r(k)$: Measured average response time in the k^{th} control period. (kT sec after the system starts).
- $e(k)$: Control error $e(k) = r(k) - R_{ref}$.
- $n(k)$: Job budget in the k^{th} control period.

Our control objective (*i.e.*, primary goal) is to guarantee that the average response time $r(k)$ converges to the set point R_{ref} within a limited *settling time*. In the meantime, we want to achieve the maximum possible job budget, $n(k)$, which is our secondary goal.

3.2 System Modeling

In order to have an effective controller design, it is important to model the dynamics of the controlled system, namely the relationship between the controlled variable and the manipulated variable. However, a well-established physical equation is usually unavailable for computer systems. Therefore, we use a standard approach called *system identification* [5] to this problem.

Based on standard qualitative analysis [6] of the controlled system, we choose to use the following difference equation to model the controlled system:

$$r(k) = \sum_{i=1}^{n_a} a_i r(k-i) + \sum_{i=1}^{n_b} b_i n(k-i) \qquad (1)$$

where n_a and n_b are the orders of the control output and control input, respectively. a_i and b_i are control parameters whose values need to be determined by system identification.

For system identification, we need to first determine the right orders for the system, *i.e.*, the values of n_a, n_b in the difference equation (1). The order values are normally a compromise between model simplicity and modeling accuracy. In this paper, we test different system orders as listed in the second and third columns in Table 1. We generate a sequence of pseudo-random digital white noise [5] as control input to stimulate the system and then measure the average response time in each control period. Based on the collected data, we use the *Least Squares Method (LSM)* to iteratively estimate the values of parameters a_i and b_i. The fourth column in Table 1 lists the estimated fits of the models in R^2, which is a standard metric in system identification to show how close the predicted data by the model is to the measured data from the real system. To verify the accuracy of the four models in different orders, we change the seed of the white noise to generate a different sequence of control input, and then compare the actual control output to those predicted by the estimated models in R^2. The results are shown in the last column of Table 1. Figure 3 shows the comparison results between the actual system output and the predicted outputs of two models. Model 1 has the lowest orders with $n_a = 0$ and $n_b = 1$ while Model 4 has the highest orders with $n_a = 1$ and $n_b = 2$. Figure 3 demonstrates that Models 1 and 4 both are sufficiently close to the actual system. We choose to use Model 1 in this paper to simplify the controller design. Therefore, our system model is:

$$r(k) = b_1 n(k-1). \qquad (2)$$

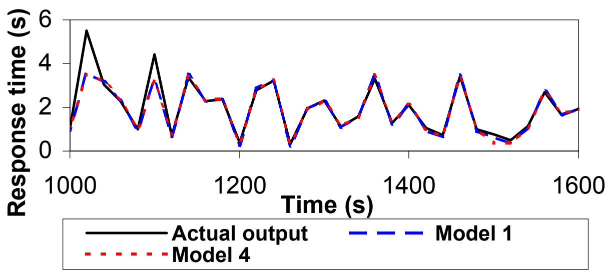


Figure 3. Comparison between predicted output and actual output

3.3 Root-Locus Design

The goal of the controller design is to meet the following requirements:

- Stability: The average response time should settle into a bounded range around the set point, R_{ref}, in response to a bounded reference input.
- Zero steady state error: The average response time should precisely settle to the set point.
- Short settling time: The system should settle to the set point within a limited time period.

Proportional-Integral (PI) control [5] has been widely adopted in industry control systems. We choose to use a PI controller because the integral part can eliminate the steady-state error. A more sophisticated PID (Proportional-Integral-Derivative) controller is not used because the derivative term may amplify the noise in response time. Following standard control theory, we design the PI controller in the Z-domain as:

$$F(z) = \frac{K_1(z - K_2)}{z - 1} \qquad (3)$$

where K_1 and K_2 are control parameters that can be analytically chosen to guarantee control performance, using standard control design methods [5]. The time-domain form of the controller (3) is:

$$n(k) = n(k-1) + K_1 e(k) - K_1 K_2 e(k-1) \qquad (4)$$

Using the Root-Locus method [5], we can choose our control parameters as $K_1 = 1/b_1$ and $K_2 = 0$ such that our closed-loop transfer function is:

$$G(z) = z^{-1} \qquad (5)$$

It is easy to prove that our controller is stable and can achieve the set point in one control period. The detailed proofs can be found in a standard control textbook [5] and are skipped due to page limitations.

4 Performance Analysis for Model Variation

Our controller is designed to achieve the control performance specified in Section 3.3 when the system model is accurate. However, in a real system, the actual system model could be different from the nominal model (2) we used to design the controller for several reasons. First, the execution time of reevaluating each subscription may be different due to different numbers of involved attributes and constraints. Second, the execution time of reevaluating the same subscription may vary at runtime as metadata updates could constantly change the attributes and constraints of publishers and consumers. Finally, the execution time may also be influenced by the hardware and software configurations of a particular system. Since developing a different controller for every different system with different sets of subscriptions is infeasible, it is important to analyze the impact of model variations on control performance.

As shown in Figure 4, an important observation from our experiments is that there exhibits an approximately linear relationship between the average response time of subscription reevaluation and the job budget despite different sets of subscriptions or systems. Based on this observation, it is valid to assume that a real system model is similar to the nominal model but with a different parameter b_1'. Without loss of generality, we model the real system as:

$$r(k) = g b_1 n(k-1) \qquad (6)$$

where $g = b_1'/b_1$ is the *execution time factor* and is used to

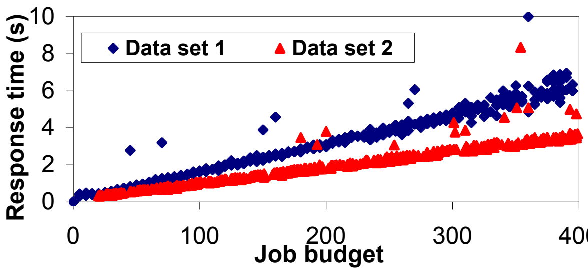


Figure 4. Linear relationship between average response time and job budgets

model the variation between the real system model (6) and the nominal model (2). Therefore, the closed-loop transfer function for a real system is:

$$G(z) = \frac{g}{z - (1-g)}. \qquad (7)$$

Now we analyze each control performance metric with the real system model.

4.1 Stability

Based on the control theory, if the system is stable, all the poles should be within the unit circle on the z-plane [5]. Based on (7), we get $|1 - g| < 1$. Hence the system remains stable as long as $0 < g < 2$. This stability range serves as an important reference when we apply our controller to different systems and subscriptions, *i.e.*, the average execution time of reevaluating a subscription in the real system should *not* be more than twice greater than the nominal value used in the controller design. This theoretical result has been verified by our extensive experiments on a physical test-bed, as presented in Section 6.4.

4.2 Steady State Error

The steady-state error of the real system can be derived as:

$$\lim_{z\to 1}(z-1)G(z)R_{ref}\frac{z}{z-1} = \lim_{z\to 1}\left(\frac{gz}{z-(1-g)}R_{ref}\right) = R_{ref}. \qquad (8)$$

Equation (8) means we are guaranteed to achieve the desired average response time as long as the system is stable.

4.3 Settling Time

We transform (7) to the time-domain:

$$r(k) = (1-g)r(k-1) + gR_{ref} \qquad (9)$$

Using the common definition of the settling time (the system settles when the output converges into the ± 0.05 range of the reference), we derive the required number of control

Table 2. Detailed test-bed configuration

	CPU	OS	JDK
Server 1	Intel Core 2 Duo Xeon 5160 3.0GHz	openSUSE 10.3 (2.6.22)	1.6.0
Server 2	AMD Athlon 64 3800+ 2.4GHz	openSUSE 10.2 (2.6.18)	1.6.0

periods, k, for the system to settle as:

$$k \geq \frac{\ln 0.05}{\ln |1-g|} \tag{10}$$

This theoretical result has also been verified by our experiments in Section 6.4.

Our above analysis gives us theoretical confidence in the performance of our controller and provides a guideline to choose the parameters in the nominal system model (2). For example, given the possible minimum and maximum values of b_1' for typical sets of subscriptions, we can choose the nominal b_1 to be a value slightly larger than $\frac{b_{1,min}'+b_{1,max}'}{2}$ such that the system is guaranteed to be stable even when the real model is unknown at design time. Similarly, given our analysis regarding settling time, we can use a smaller b_1 for faster reaction to variations or a greater b_1 to reduce the system sensitivity. This kind of theoretical guidance is in sharp contrast to commonly used heuristic solutions that heavily rely on extensive empirical evaluation and manual tuning for desired system response.

5 System Implementation

Our test-bed includes two servers that are detailed in Table 2. The two servers are connected via a network switch. We use the aforementioned INFOD system as a representative real-time information dissemination system to implement our feedback control loop. The INFOD registry is implemented in Oracle Database 11.1g on Server 1 to run the metadata matching process. Since our objective is to evaluate the performance of the control loop running on Server 1, all the INFOD publishers, consumers and subscribers are implemented on Server 2 to simplify experimental setup. They communicate with the registry through JDBC (Java DataBase Connectivity), which is a Java package enabling Java programs to execute SQL statements in databases.

The process of metadata matching is implemented as an Oracle PL/SQL procedure and configured as an event-based task. After a metadata update is received from the publishers and consumers, it is compared with selected subscriptions to check whether the subscriptions need to be reevaluated because of this update. If reevaluation is necessary, the subscriptions will be enqueued into a system queue called SCHDL_QUEUE. The Oracle database scheduler allocates a job slave process, from the job slaves pool shown in Figure 2, to handle each of the incurred metadata matching processes. Metadata matching is configured to have a priority of 3 out of the five priorities (1 to 5, with 1 as the highest) provided by the Oracle database scheduler.

The control period is selected based on a trade-off between sensitivity to system noise and reaction speed of the controller. On one hand, each control period should be long enough to include multiple metadata updates such that the influence of system disturbance and noise incurred to the execution time of each single update can be reduced. On the other hand, a longer control period leads to slower reaction to workload variations. In our experiments, the interarrival intervals of incoming metadata updates are within a range of $(2s, 5s)$. Therefore, the control period is set as $20s$ to include at least 4 updates. The measured average overhead of running the controller and monitor is about $20ms$, roughly 0.1% of a control period. This overhead should be accpetable to most systems.

We now introduce the implementation details of each component in our control loop. Note that the control loop is assigned the highest priority such that it can run periodically even when the system becomes overloaded.

Monitor: The monitor is invoked at the end of each control period. It calculates the average response time $r(k)$ of all the subscriptions reevaluated in this control period based on the view of DBA_SCHEDULER_JOB_RUN_DETAILS in the database. The sampled average response time is then sent to the controller.

Controller: Based on system identification introduced in Section 3, we get the following model for the INFOD system:

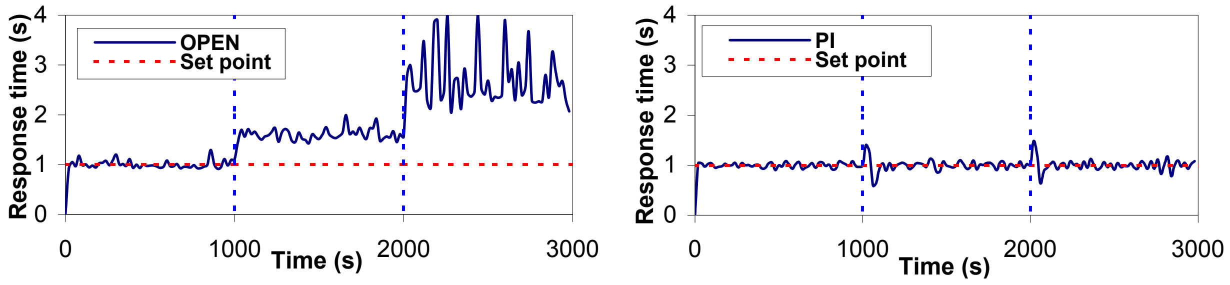
$$r(k) = 0.009055n(k-1) \tag{11}$$

Based on the model, the PI controller used in our experiments is:

$$n(k) = n(k-1) + 110.436e(k) \tag{12}$$

where $e(k)$ is the difference between the measured average response time and the set point. $n(k-1)$ is the job budget in the last control period. The controller calculates the desired job budget $n(k)$ and sends it to the scheduler for enforcement.

Scheduler: The scheduler is an application-level Oracle PL/SQL trigger (similar to a procedure) implemented on the top of the original scheduler in the Oracle database. Oracle provides the DBMS_SCHEDULER package from which we can call a collection of scheduling functions and procedures to manage jobs (*e.g.*, create, configure and schedule) [7]. The application-level scheduler monitors the metadata tables of publishers and consumers, and serves as an actuator for controlling the average response time of metadata matching. Upon the arrival of metadata updates, the scheduler reevaluates all high-priority subscriptions and certain number of low-priority subscriptions based on the job budget received from the controller.



(a) OPEN (b) PI

Figure 5. Control accuracy comparison between open-loop system and PI controller

6 Experimentation

In this section, we present the experimental results conducted on the test-bed introduced in Section 5. We first introduce two baselines and compare them with the designed PI controller, respectively, in terms of control accuracy and system quality of service. We then verify the theoretical results derived in Section 4 by examining the performance of our controller with different execution time factors.

In all the experiments, we use $1s$ as the set point for the response time of metadata matching (*i.e.*, subscription reevaluation). The same set of metadata updates are used with interarrival intervals randomly distributed within $[2s, 5s]$ if not otherwise indicated.

6.1 Baselines

We use two algorithms: OPEN and Ad Hoc, as our baselines. OPEN is an open-loop algorithm that uses fixed job budget based on the estimated execution time of metadata matching. Although OPEN can result in desired average response time when estimated execution time is accurate, it may violate the timing constraint of metadata matching when execution time is underestimated. The resultant violation of time constraint may cause highly undesired accidents in real systems. At the same time, it may also cause the average response time unnecessarily shorter than the desired value when execution time is overestimated. Consequently, the job budget includes fewer low-priority subscriptions, which may lead to poor system quality of service.

Ad Hoc is a heuristic-based adaptive controller that represents a typical solution to response time control. At each control invocation, it simply raises or lowers the job budget by a certain step, depending on whether the measured average response time is lower or higher than the set point. However, it is commonly difficult for Ad Hoc to decide the best step size in a real system. Small step size may cause undesired slow response while large step size may cause the system to oscillate dramatically. In our experiments, Ad Hoc has a control period of 20s just as the PI controller.

6.2 Comparison with OPEN

In this subsection, we compare PI with OPEN in terms of control accuracy and system quality of service.

6.2.1 Control Accuracy

In this experiment, we test both OPEN and PI in a scenario common to many real-time information dissemination systems. In this scenario, the average reevaluation time of subscriptions has a sharp increase at run time. This is common to real-time information dissemination systems and could be caused by several reasons such as metadata updates that add new publishers or consumers to the system. We first adopt the same set of subscriptions that are used to design OPEN and PI. As a result, the estimated reevaluation time of a subscription is accurate at the beginning for both of the two controllers, *i.e.*, $g = 1$. At time 1000s, the reevaluation time has a sharp increase and becomes 1.4 times greater than the estimation, *i.e.*, $g = 1.4$. At time 2000s, more updates further increase the real execution time to have $g = 1.8$.

As discussed before, OPEN is designed to have a fixed job budget to achieve the desired response time based on accurate knowledge of the subscription reevaluation time. Figures 5(a) shows that OPEN indeed achieves the desired response time at the beginning. However, OPEN violates the response time constraint after the execution time increase because OPEN cannot adapt the system for desired response time by reducing the job budget. Consequently, the average response time of metadata matching increases to $1.6s$. This long delay is highly undesired in real-time systems because the consumers of high-priority subscriptions (*e.g.*, firefighters) may therefore get into dangerous situations when they fail to receive important information (*e.g.*, periodic fire condition reports) within the required time frame. With a further increase of execution time at time $2000s$, OPEN has even longer delay and also causes the system to have large oscillations. This is because the execution time of reevaluating those subscriptions becomes longer than the interarrival interval of metadata updates. As a result, the system cannot complete the subscription reevaluation within one interval, which leads to jitters in different control periods.

In contrast to OPEN, as shown in Figure 5(b), PI

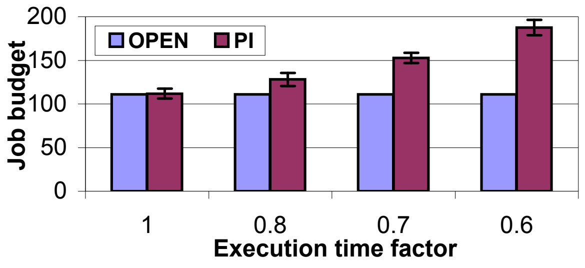


Figure 6. System quality of service comparison between OPEN and PI under different execution time factors

achieves the desired response time in the same scenario, despite the significant variations in the subscription reevaluation time. This is because PI monitors the real response time and dynamically adapts the job budget based on control theory. This experiment demonstrates that OPEN only works well when we have accurate knowledge of the average execution time of subscription reevaluation, and the average execution time never changes. In contrast, our PI controller can guarantee the average response time converges to the set point in spite of significant average execution time variations.

6.2.2 System Quality of Service

The primary goal of our feedback control architecture is to guarantee that the average response time of metadata matching (*i.e.*, subscription reevaluation) is shorter than a real time constraint. The secondary goal is to have a job budget (*i.e.*, total number of reevaluated subscriptions) as large as possible in each control period to achieve the best possible system quality of service. Because OPEN violates the response time constraint when the execution time factor, g, is larger than 1, as shown in Figure 5(a), we only consider the situations when $g \leq 1$ in the comparison between PI and OPEN. Specifically, we use four sets of subscriptions with the execution time factor as 1, 0.8, 0.7 and 0.6, respectively. The resultant job budget for each subscription set is shown in Figure 6. Each data point is the average of 50 runs in the steady state for each execution time factor. Since OPEN uses a fixed job budget of 111, it has an unnecessarily short average response time when $g < 1$. In contrast, PI can control the average response time to converge to the desired set point by dynamically increasing the job budget from 111.9 to 128.1, 152.7 and 187.6. Although OPEN and PI both meet the time constraint, PI can reevaluate more low-priority subscriptions thus provides better system quality of service.

6.3 Comparison with Ad Hoc

In this subsection, we compare PI with Ad Hoc in terms of control accuracy and system quality of service.

6.3.1 Control Accuracy

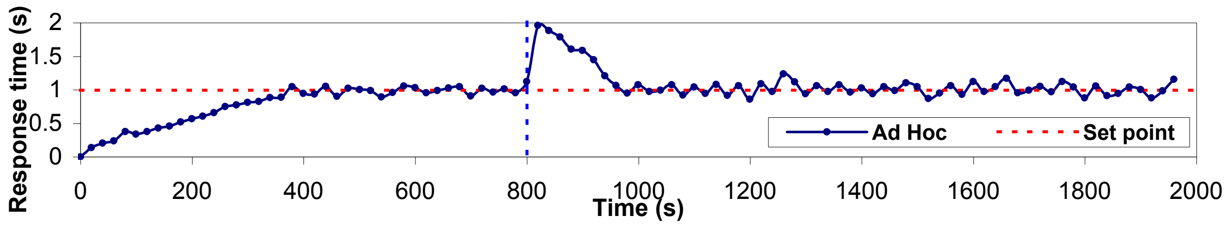
In this experiment, Ad Hoc is designed to achieve the best performance with a certain set of subscriptions whose real reevaluation time is smaller than the estimation ($g = 0.6$). Based on extensive tuning and testing which are common to heuristic-based solutions, we find that a step size of 10 for job budget change gives Ad Hoc the best performance.

Figures 7(a) and 7(b) show the results of Ad Hoc and PI, respectively. At the beginning, Ad Hoc increases the job budget by 10 in each control period until the average response time reaches the set point. Since Ad Hoc relies on a fixed step to reach its steady state, it takes Ad Hoc 19 control periods ($380s$) to settle down to the set point. In contrast, PI can effectively adapt its step size based on control theory. As a result, PI only takes 5 control periods ($100s$) to enter its steady state. Please note although it is possible to configure Ad Hoc with a greater step size for faster settling time, doing so may cause Ad Hoc to have large oscillation in steady state, which is undesired for system performance. In the steady state until $800s$, the two controllers work similarly with almost the same deviations (0.055 for Ad Hoc and 0.051 for PI).

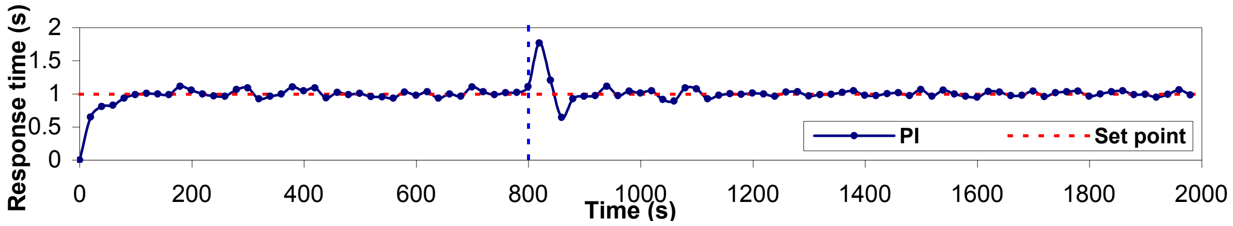
At time $800s$, we assume the execution time of subscription reevaluation increases (with $g = 1$) due to the reasons given in Section 6.2.1. Similarly, Ad Hoc takes 10 control periods ($200s$) to return to the steady state. In contrast, PI only takes 5 control periods. Since the step size of Ad Hoc is designed to achieve the best performance when the execution time factor is 0.6, the response time under Ad Hoc oscillates around the set point and thus frequently violates the time constraint. In contrast to Ad Hoc, PI converges to the set point smoothly with a small deviation of 0.044. This experiment demonstrates that Ad Hoc cannot effectively adapt to varying execution time. It is commonly difficult for Ad Hoc to tune its step size at runtime, due to the lack of established adaptation methods. In contrast, PI relies on control theory for adaptation and thus can provide response time guarantees in spite of execution time variations.

6.3.2 System Quality of Service

In this experiment, we show PI also has better system quality of service than Ad Hoc. We use four sets of subscriptions with different execution time factors (0.6, 1.0, 1.4 and 1.8). As discussed before, it is commonly difficult for Ad Hoc to have a step size that is good for all workloads, because there exists a trade-off between system oscillation and settling time. Finer step size usually leads to smaller oscillation in steady state, but may cause longer settling time and so slower response to workload variations. Long settling time could be dangerous to the consumers of high-priority subscriptions (*e.g.*, firefighters), when the system needs to quickly converge back to the desired response time from an unexpected workload increase. Therefore, to have an acceptable settling time, we tune the step size of Ad Hoc such that Ad Hoc has a settling time within 10 control peri-



(a) Ad Hoc



(b) PI

Figure 7. Control accuracy comparison between Ad Hoc and PI

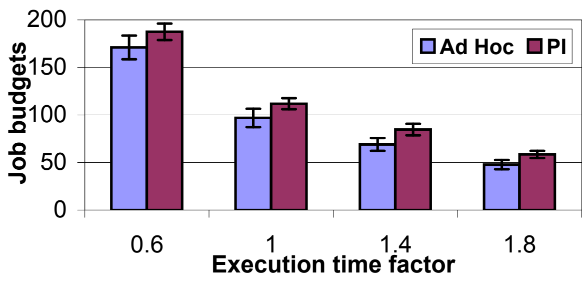


Figure 8. System quality of service comparison between Ad Hoc and PI under different execution time factors

ods ($200s$). We do this for every subscription set to have a fair comparison with PI. However, this shorter settling time causes Ad Hoc to have larger system oscillation and thus frequent violations of time constraint. Hence, to achieve similar performance with PI for a fair comparison, a safe margin needs to be used for Ad Hoc to lower its set point a little for fewer violations of constraint. We calculate the safe margin as follows. First, we tune a step in the way we describe above for each set of subscriptions. We then calculate the standard deviation of a typical run for each set of subscriptions with its individual step. Finally, we use the maximum deviation of the four sets of subscriptions as the safe margin for Ad Hoc. The safe margin is calculated as $0.08s$ in our experiments. We then compare Ad Hoc and PI for the average job budget of 50 control periods in steady state. The results shown in Figure 8 demonstrate that PI can reevaluate more low-priority subscriptions, thus outperforms Ad Hoc in term of system quality of service.

6.4 Control Performance under Different Execution Time Factors

In this subsection, we verify the theoretical performance analysis of PI with different execution time factors.

6.4.1 System Stability

As discussed in Section 4, the stability of our controller is related to the execution time factor g. In this experiment, we verify that theoretical result by using experiments to test the system stability with different g. In Figure 5(b), the deviation of response time after the controller settles down to the steady state slightly increases from 0.047 to 0.064 and 0.078 as the execution time factor increases from 1 to 1.4 and 1.8 at time $1000s$ and $2000s$, respectively. However, when the execution time factor changes to 2.6, as shown in Figure 9, the controller becomes unstable as the response time oscillates significantly between 0 and a large value. The result is consistent with the stability range ($0 < g < 2$) derived in our stability analysis in Section 4.

To further investigate the relationship between the system stability and g, we plot the mean and deviation of the average response time with different execution factors in Figure 10. In the figure, each data point is the mean of 50 runs when the controller is in its steady state. The standard deviation of the average response time indicates the intensity of oscillation. As the execution time factor increases from 0.6 to 1.8, the standard deviation remains below 0.1. This small deviation is caused by uncertainties in computer systems (*e.g.*, cache, pipelining). When the execution time factor increases to 1.95 that is close to the theoretical stability bound, the standard deviation starts to increase noticeably. When the execution time factor locates outside the theoretical stability range, the system becomes unstable as it has significant oscillation and the average response time deviates from the set point. Figure 10 shows that the empirical results on a physical test-bed validate our theoretical analysis. This experiment demonstrates that, in contrast to OPEN and Ad Hoc which rely on exhaustive iterations of tuning and testing, PI can provide theoretical guarantee that system is stable within a certain range of g.

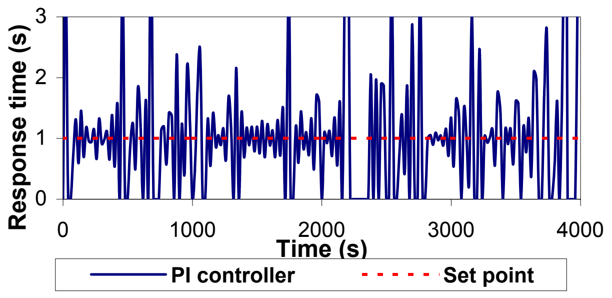


Figure 9. Controller outputs when g=2.6

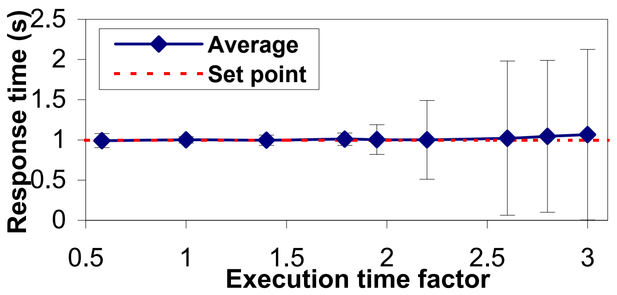


Figure 10. Average response time and deviation under different execution time factors

6.4.2 Settling Time

In Section 4, our theoretical analysis (Equation (9)) shows that the settling time of the PI controller is also related to the execution time factor g. In this experiment, we verify this result with different sets of subscriptions with the execution time factor as 0.6, 0.8, 1.0, 1.2, 1.4 and 1.7, respectively. For each set of subscriptions, we measure the average settling time as the number of control periods needed to enter steady state. Figure 11 shows the average and deviation of settling time in 10 repeated runs for different execution time factors. The theoretical values are rounded integers from the calculated results using Equation (9) because the number of control periods cannot be fractional. As shown in the figure, the closer the execution time factor approaches 1, the shorter the settling time is. The largest deviation is 1.23 when $g = 1.7$. We can see that the experimental results are very close to the theoretical results. This experiment validates our theoretical analysis.

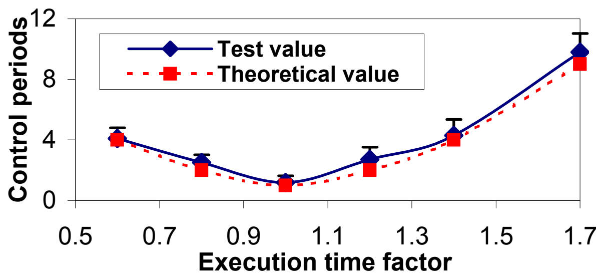


Figure 11. Settling time under different execution time factors

7 Related Work

Control-based real-time data matching was originally motivated by the INFOD project [2]. The INFOD scheduling mechanism use heuristics for adaptation that do not provide any performance guarantees, but are instead evaluated empirically. In sharp contrast, control-based approach can provide robust and analytic performance guarantees that are crucial in real-time information dissemination systems.

Control theoretic approaches have been applied to a number of computing and networking systems. A survey of feedback performance control for software is presented in [8]. A number of feedback-based real-time processor scheduling algorithms(*e.g.*, [9] [10] [11] [12]) have also been presented in the literature. These algorithms only controlled the allocation of the computing resource on a server, and do not address average response time in information dissemination systems. Control theory has also been applied to design and analyze data services [13], power management [14] [15] and Internet servers [16][17]. This paper is different from these control-based solutions in that it aims to support real-time information dissemination.

Some related work has been done to develop real-time information dissemination system. For example, real-time data broadcasting has been addressed in [18] and [19]. Hu has presented an on-demand time-critical data broadcast scheme for asymmetric wireless networks [20]. Xu et al. have presented an on-demand broadcast scheduling algorithms that takes into account the time constraints [21]. However, those related projects mainly focus on broadcasting in wireless data dissemination systems with asymmetric links. This paper is different from the related work because we try to address a different but equally important problem: real-time metadata matching, which is the bottleneck of many existing real-time information dissemination systems such as INFOD. Real-time metadata matching is crucial to information dissemination because information can be disseminated from publishers to matched consumers only when the matching results are generated from the metadata matching process.

Some other prior work has been done on Quality of Service management in databases. For example, some on-demand updating algorithms have been developed to skip unnecessary updates and allow better CPU utilization [22] [23] [24]. Kang et al. have presented feedback controllers to manage the deadline miss ratio and sensor data freshness [25]. Amirijoo et al. have presented feedback controllers for QoS management using imprecise computations [26]. Haritsa et al. have presented value-based Scheduling algorithms in real-time databases [27]. However, most prior work is based on the assumption that transactions are either periodic sensor updates or aperiodic user transactions, and then adapt the number of updates for desired CPU utilization. All those methods cannot be directly applied to information dissemination systems because: (1) metadata matching is triggered by metadata updates, which is different from the scenario mentioned in prior papers; (2) the execution time of metadata updates is trivial compared with that of data matching which accounts for more than 90% of CPU time. Therefore, adjusting the number of metadata updates is not sufficient; (3) instead of emphasizing a hard deadline on each individual user transaction, it is more important to control statistical performance metrics such as average response time in real-time information dissemination.

8 Conclusion

In this paper, we presented a feedback controller to adaptively meet the response time constraints on metadata matching in an example information dissemination system. Our controller features a rigorous design based on well-established feedback control theory for guaranteed control accuracy and system stability. We verify our theoretical analysis with extensive experiments on a physical test-bed. Our empirical results also demonstrate that our controller outperforms an open-loop solution and a typical heuristic solution, by having more accurate control and better system quality of service.

Acknowledgements

This research is supported in part by a subcontract from the Department of Homeland Security-sponsored Southeast Region Research Initiative (SERRI) at the National Nuclear Security Administration's Y-12 National Security Complex, and by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory under Contract No. DE-AC05-00OR22725. We would also like to thank Dieter Gawlick at Oracle Corporation for his help with the INFOD system.

References

[1] F. Hayes-Roth, " Model-Based Communication Networks and VIRT: Orders of Magnitude Better for Information Superiority," *Military Communications Conference, 2006. MILCOM 2006*, pp. 1–7, Oct. 2006.

[2] INFOD-WG, *Information Dissemination in the Grid Environment - Base Specifications*, Open Grid Forum (2004-2007), May 2007.

[3] X. Wang and M. Chen, "Cluster-level feedback power control for performance optimization," in *Proc. of the 14th International Symposium on High-Performance Computer Architecture*, Salt Lake City, UT, Feb. 2008.

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

[5] G. F. Franklin, J. D. Powell, and M. Workman, *Digital Control of Dynamic Systems, 3rd edition*. Addition-Wesley, 1997.

[6] M. Verhaegen and V. Verdult, *Filtering and System Identification, A Least Square Approach*. Cambridge University Press, 2007.

[7] D. Raphaely, *Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1.)*, Oracle, Sep. 2007.

[8] T. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback performance control in sofware services," *IEEE Control Systems*, vol. 23, no. 3, Jun. 2003.

[9] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole, "Analysis of a reservation-based feedback scheduler," in *IEEE RTSS*, Dec. 2002.

[10] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son, "Feedback control real-time scheduling: Framework, modeling, and algorithms," *Journal of Real-Time Systems*, vol. 23, no. 1/2, pp. 85–126, Jul. 2002.

[11] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-rate scheduling," in *OSDI*, 1999.

[12] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Arzen, "Feedback-feedforward scheduling of control tasks," *Real-Time Systems*, vol. 23, no. 1, pp. 25–53, Jul. 2002.

[13] M. Amirijoo, N. Chaufette, J. Hansson, S. H. Son, and S. Gunnarsson, "Generalized performance management of multi-class real-time imprecise data services." in *IEEE RTSS*, 2005.

[14] C. Lefurgy, X. Wang, and M. Ware, "Server-level power control," in *Proceedings of the 4th IEEE International Conference on Autonomic Computing (ICAC)*, 2007.

[15] K. Skadron, T. Abdelzaher, and M. R. Stan, "Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management," in *HPCA '02: Proceedings of the 8th International Symposium on High-Performance Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2002, p. 17.

[16] Y. Diao, J. L. Hellerstein, S. Parekh, H. Shaikh, and M. Surendra, "Controlling quality of service in multi-tier web applications," in *ICDCS*, 2006.

[17] R. Zhang, C. Lu, T. F. Abdelzaher, and J. A. Stankovic, "ControlWare: A Middleware Architecture for Feedback Control of Software Performance," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*. Vienna, Austria: IEEE, Jul. 2002.

[18] Y.-C. Chung, C.-C. Chen, and C. Lee, "Time-constrained service on air," in *25th IEEE International Conference on Distributed Computing Systems (ICDCS 2005)*, Columbus, Ohio, 2005.

[19] J. Fernandez-Conde and K. Ramamritham, "Adaptive dissemination of data in time-critical asymmetric communication environments," *Mobile Networks and Applications*, vol. 9, no. 5, pp. 491 – 505, Oct. 2004.

[20] C.-L. Hu, "On-demand real-time information dissemination: A general approach with fairness, productivity and urgency," in *Proceedings of the 21st International Conference on Advanced Networking and Applications (AINA07)*, Niagara Falls, Canada, May 2007.

[21] J. Xu, X. Tang, and W.-C. Lee, "Time-critical on-demand data broadcast: Algorithms, analysis, and performance evaluation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 1, pp. 3–14, Jan. 2006.

[22] Q. N.Ahmed and S. V.Vrbsky, "Triggered Updates for Temporal Consistency in Real-TimeDatabases," *Real-Time Systems*, vol. 19, no. 3, pp. 209 – 243, Nov. 2004.

[23] T. Gustafsson and J. Hansson, "Dynamic on-demand updating of data in real-time database systems," in *Proc. of the 19th Annual ACM Symposium on Applied Computing*, Nicosia, Cyprus, Mar. 2004.

[24] T. Gustafsson and J.Hansson, "Data management in realtime systems: a case of on-demand updates in vehicle control systems," in *Proceedings of the Real-Time Application Symposium (RTAS 2004)*, Toronto, Canada, May 2004.

[25] K. Kang, S. Son, and J. Stankovic, "Managing deadline miss ratio and sensor data freshness in real-time databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 1200 – 1216, Jul. 2004.

[26] M. Amirijoo, J.Hansson, and S.Son, "Specification and management of qos in real-time database supporting imprecise computations," *IEEE Transactions on Computers*, vol. 55, no. 3, pp. 304–319, Mar. 2006.

[27] J. R. Haritsa, M. J. Carey, and M. Livny, "Value-based scheduling in real-time database systems," *VLDB Journal: Very Large Data Bases*, vol. 2, no. 2, pp. 117–152, 1993.