Integrated Control of Matching Delay and CPU Utilization in Information Dissemination Systems

Ming Chen, Xiaorui Wang, and Ben Taylor
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN 37996
{*mchen11, xwang, btaylo22*}*@utk.edu*

***Abstract*— The demand for high performance information dissemination is increasing in many applications, such as e-commerce and security alerting systems. These applications usually require that the desired information be matched between sources and sinks based on established subscriptions in a timely manner while a maximal system throughput be achieved to find more matched results. Existing work primarily focuses on only one of the two requirements, either timeliness or throughput. This can lead to an unnecessarily underutilized system or poor guarantees on matching delays. In this paper, we propose an integrated solution that controls both the matching delay and CPU utilization in information dissemination systems to achieve bounded matching delay for high-priority information and maximized system throughput in an example information dissemination system. Our solution is based on optimal control theory for guaranteed control accuracy and system stability. Empirical results on a physical testbed demonstrate that our controllers can guarantee the timeliness requirements while achieving maximized system throughput.**

I. Introduction

During the last decade, information dissemination has started to play a critical role in the design and development of a large class of applications. For example, 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 security alerting systems, threats detected by various sensors must be reported to the appropriate authorities within certain time constraint. In these applications, matches between numerous (*e.g.*, thousands of) sources and numerous sinks should be found accurately, efficiently, and more importantly, in a timely manner. These requirements have been generally described as *Valuable Information at the Right Time (VIRT)* [1]. This emphasizes that consumers of information should receive the accurate information that is of interest to them as soon as it is available or whenever it is requested. In the meantime, a maximal possible system throughput should be achieved to find more matched results between data sources and sinks.

INFOD (INFOrmation Dissemination) [2] is an example information dissemination system that aims to support timely delivery of valuable information for a wide range of applications. As shown in Figure 1, information sources and sinks are defined as *publishers* and *consumers*, respectively. *Subscriptions* are prescribed requests of information and are submitted by *subscribers* on behalf of consumers. Publishers, consumers, and subscribers advertise their attributes and constraints, which are generally referred to as *metadata*, in a database called *registry*. For example, a consumer may have its location as an attribute and have a constraint on desired publishers: they must be located within 10 miles. An example subscription is: all sensors (publishers) send their traffic information to all drivers (consumers) within 10 miles no later than 5 seconds after a jam occurs. Subscriptions may have different priorities. For example, subscriptions of traffic information from police should have a higher priority than those from ordinary drivers. Metadata can be updated periodically and aperiodically. INFOD needs to find new matches between the metadata of publishers and consumers based on the subscriptions within a time constraint when a metadata update arrives, which we refer to as *subscription reevaluation* or *metadata matching*. Meanwhile, more matched results are desired by reevaluating the maximal number of subscriptions, which means that the registry server should be efficiently utilized. Based on the matched results, publishers are informed where to send filtered information. The information is then sent to the matched consumers without passing through INFOD.

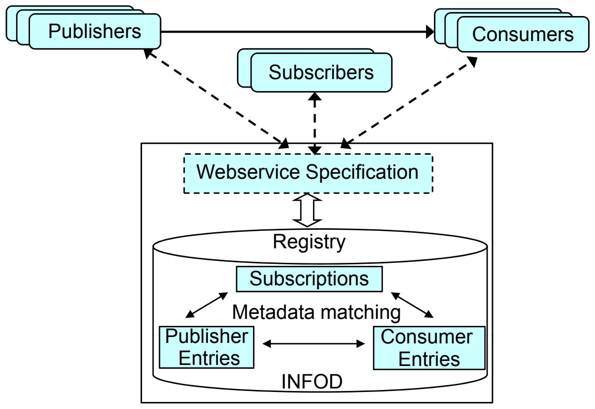


Fig. 1. INFOD: an example information dissemination system

To guarantee both the bounded matching delay and efficient system utilization in information dissemination systems faces two major challenges. *First*, when an update arrives, the system may need to reevaluate all related subscriptions to find new matching results between the publishers and consumers. For example, a driver may constantly update its location attribute. As a result, all related subscriptions in the registry need to be continuously reevaluated by rerunning metadata matching to ensure that the driver receives information from the right locations. However, given the large number of publishers and consumers, reevaluating all related subscriptions may cause severe system overload and unacceptably long delays. Therefore, to guarantee bounded matching delay and avoid system overload, only a few of the subscriptions can be picked up for reevaluation. *Second*, metadata updates may arrive at unpredictable intervals. Given a constant number of subscription reevaluations, the registry may suffer either over or underutilization, depending on whether the interval is short or long. Overutilization may lead to unbounded matching delay, while underutilization may lead to unnecessarily low system throughput, resulting in incomplete matched results. Both unbounded matching delays and unnecessarily underutilized systems are undesirable. For example, a driver may fail in finding the fastest route, either due to a late notification of traffic information, or because the registry fails in finding the existing matched traffic information.

To address all the challenges, we propose a mechanism by applying a batching window to group those updates at small interarrival intervals and release them all together. We differentiate subscriptions by two priorities: high and low. After each release, all high-priority subscriptions are reevaluated first, and then a certain number of low-priority subscriptions are selected for reevaluation. The low-priority subscriptions are reevaluated in a round-robin way. Clearly, both the batching window size and number of subscriptions chosen to be reevaluated affect the matching delay and utilization of the registry server. Therefore, it is important to determine the batching window size and number of subscriptions in an integrated way such that the average matching delay of all high-priority subscriptions are guaranteed to meet the specified constraint, and the registry server can be efficiently utilized to achieve maximal system throughput.

In this paper, we present a novel solution to control both the matching delay of the high-priority subscriptions and CPU utilization of the registry server in an integrated manner. Under our control solution, the average matching delay of all the high-priority subscriptions can converge to the specified constraint, while the CPU utilization is controlled to a desired set point so that the system can reevaluate the maximized number of subscriptions. Specifically, the contributions of our work are four-fold:

- We propose a novel batching mechanism to address the challenges of metadata matching in information dissemination systems;
- We model the average matching delay and CPU utilization of the registry server and validate the model with data measured on a physical testbed.
- We design an integrated control solution based on the model to guarantee the timeliness of all the high-priority subscriptions and system throughputs simultaneously and present a detailed analysis of system stability based on optimal control theory;
- We implement our control solution based on a real information dissemination system and present empirical results on a physical testbed to demonstrate that our solution can guarantee the timeliness for all high-priority subscriptions and achieve maximized system throughputs.

The rest of the paper is organized as follows. Section II discusses related work. Section III introduces the overall architecture of the system. Section IV presents the system modeling, controller design and analysis. Section V describes the implementation details of each component in the system architecture. Section VI presents the results of our experiments and Section VII concludes the paper.

II. Related Work

Some related work has been done to guarantee the average response time and CPU utilization in computing systems. For example, Horvath et al. address End-to-End response time control in multi-tier web servers by using dynamic voltage scaling [3]. Wang et al. present a load balancing controller to control the relative response time among virtualized servers [4]. Chen et al. propose a hierarchical control architecture to guarantee the deadline of power grid computing tasks by controlling CPU utilization [5]. Diao et al. develop MIMO control algorithms to control the processor and memory utilization for Apache web servers [6]. However, most of the work only concerns either the average response time or CPU utilization, but not both of them. This paper highlights the advantages of controlling both the average matching delay of high-priority subscriptions and CPU utilization simultaneously to achieve both bounded delay and maximized system throughputs.

Our previous work [7] presents an initial solution to control only the processing delay of metadata matching. This paper is significantly different because: 1) we design a batching mechanism that can handle unpredictable update interarrival intervals; 2) we simultaneously control matching delay (including waiting time and processing time) and CPU utilization, based on the optimal control theory, for better system throughputs. We use our previous work as a baseline in our evaluation.

Information dissemination has recently received a lot of attention. For example, Bullet, a multi-receiver data dissemination mesh that aims for the high-bandwidth data dissemination for large-scale distributed systems, has been presented by Kostić et al. [8]. Voulgaris et al. propose a self-organizing content-based publisher/subscriber system for dynamic large-scale collaborative networks [9]. Iamnitchi et al. develop an interest-aware information dissemination system for small-world communities [10]. However, these related algorithms primarily attempt to improve the efficiency and accuracy of information dissemination. This paper is different from them because we focus on a different, but equally important, problem, *i.e.*, guaranteeing bounded delay while achieving maximized system throughput, from the view of the system level.

Batch and priority-based algorithms has been applied in a number of other computing systems, *e.g.*, scheduling for clusters [11][12], memory controllers [13], and network switching [14][15]. However, all these algorithms are designed for specific scenarios with different goals, such as performance/engergy optimization, and decreased packet loss rate. To our best knowledge, those algorithms cannot be applied in information dissemination systems like INFOD due to different scenarios and goals.

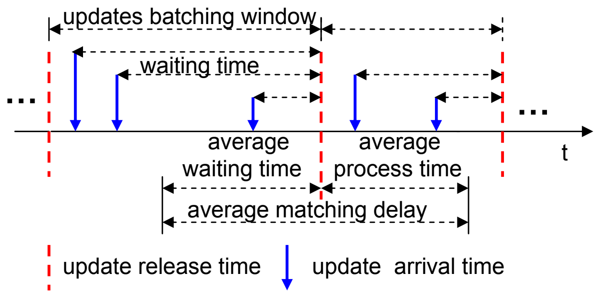


Fig. 2. Illustration of batching window

Some other previous work has been done on Quality of Service management in databases. For example, some on-demand updating algorithms have been developed to improve CPU utilization by skipping unnecessary updates [16][17][18]. Kang et al. have presented feedback controllers to manage the deadline miss ratio and sensor data freshness [19]. Amirijoo et al. have presented feedback controllers for QoS management using imprecise computations [20]. Haritsa et al. have presented value-based Scheduling algorithms in real-time databases [21]. However, those studies are based on the assumption that workload is either periodic sensor updates or aperiodic user transactions, and then adapt the number of updates for desired CPU utilization. All these methods cannot be directly applied to information dissemination systems.

III. System Overview

In this section, we first introduce the batching mechanism used in our solution, and then give a high-level description of the control architecture.

A. Batching Mechanism

As shown in Figure 2, we employ a *batching window* to accumulate all updates that come within the window and release them in a batch. At the end of each batching window, defined as the *releasing time*, all high-priority subscriptions are chosen for reevaluation first, and then some of low-priority subscriptions are reevaluated. All the low-priority subscriptions are picked up in a round-robin way. Specifically, we refer to the length of time between two adjacent releasing times as the *batching window size*. We define the difference between the arrival time and the releasing time of an updateas as the *waiting time* of that update. The difference between the releasing time and the time when a subscription is completed is defined as the *processing time* of that subscription. The *average matching delay* of all high-priority subscriptions is calculated as the sum of the average waiting time of all updates in a batching window and the following average processing time of all reevaluated high-priority subscriptions after the batching window.

In this paper, the number of all reevaluated subscriptions each second is defined as the ***overall throughput*** of the system. The number of all subscriptions that are reevaluated after each batching window is defined as the ***job budget***, which should be maximized so that more low-priority subscriptions can be reevaluated after each batching window. The number of reevaluated low-priority subscriptions each second is defined as the ***throughput of low-priority subscriptions***. Hereinafter, we refer to both the overall throughput and the throughput of low-priority subscriptions as ***system throughputs***. In addition, the average time that all low-priority subscriptions take to be reevaluated at least once is defined as the ***average matching interval***.

B. Control Architecture

In this subsection, we provide a high-level description of the integrated control loop that controls the average matching delay and CPU utilization by dynamically adjusting the batching window size and job budget.

As shown in Figure 3, the key components in the integrated control loop include the integrated controller, delay and CPU utilization monitor, updates batcher, and scheduler. The control loop is invoked periodically at the end of every control period as follows:

1) The monitor measures the average matching delay and CPU utilization in the last control period, and sends the values to the controller. The average matching delay and CPU utilization are the *controlled variables* in the control loop;
2) The controller calculates the appropriate job budget and batching window size for the next control period based on the differences between the set points and measured controlled variables. The job budget and batching window size are the *manipulated variables* in the control loop;
3) Based on the calculated batching window size, all updates that arrive within the window are released in a batch. At the releasing time, the batcher calculates the average waiting time of all updates in the window and notifies the scheduler in the database;
4) After receiving the notification from the batcher, the scheduler, based on the calculated job budget, schedules all of the related high-priority subscriptions and the desired number of low-priority subscriptions to be reevaluated in the system.

Since the core of the control loop is the controller, we introduce the design and analysis of the controller in the next section.

IV. Integrated Controller

In this section, we present the problem formulation, modeling, design, and analysis of the integrated controller.

A. Problem Formulation

We first introduce the following notation.

- $\mathbf{R_{ef}}$: The reference vector, $\mathbf{R_{ef}} = \begin{bmatrix} R_d & R_u \end{bmatrix}^T$, where R_d and R_u are the reference values of the average matching delay and CPU utilization, respectively.
- T: The control period of the integrated controller.
- $w(k)$, $p(k)$, and $d(k)$: The average waiting time of each update, the average processing time of the high-priority subscriptions, and the average matching delay in the k^{th}

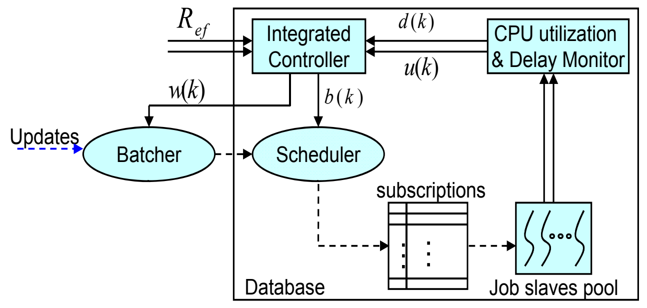


Fig. 3. Integrated control architecture

control period, respectively. Specifically, $d(k) = p(k) + w(k)$.

- $u(k)$, $s(k)$, and $b(k)$: The average CPU utilization, the batching window size, and job budget, respectively, in the k^{th} control period.
- $\overline{d}$, $\overline{u}$, $\overline{s}$ and $\overline{b}$: The operating points of $d(k)$, $u(k)$, $s(k)$, and $b(k)$.

The goal of the integrated control is to simultaneously control both the average matching delay of the high-priority subscriptions and CPU utilization to their respective set points, R_d and R_u. The purpose of controlling the average matching delay is to guarantee that all matched results of high-priority subscriptions can be disseminated within a time constraint after a metadata update arrives. The purpose of controlling the CPU utilization is to achieve maximized system throughputs (*i.e.*, the overall throughput and throughput for low-priority subscriptions) so that more subscriptions can be reevaluated to find more matched results between data sources and sinks. The selection of the set point of the CPU utilization is a compromise between the control accuracy of the average matching delay and system throughputs. The higher the CPU utilization, the larger the system throughputs, but the more difficult it is to control the average matching delay accurately, which may result in undesired large oscillations.

In this paper, the integrated controller is formulated as an optimal control problem to find the optimal batching window size and maximize the job budget while controlling matching delay and CPU utilization to their set points.

B. 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 variables (*i.e.*, $d(k)$ and $u(k)$) and the manipulated variables (*i.e.*, $b(k)$ and $s(k)$). However, a well-established physical equation is usually unavailable for computing systems. Therefore, we apply a standard approach called system identification [22] to this problem.

We use the following difference equation to model the controlled system [23]:

$$\mathbf{y(k)} = \sum_{i=1}^{n_a} \mathbf{A_i y(k-i)} + \sum_{i=1}^{n_b} \mathbf{B_i v(k-i)}, \tag{1}$$

where $\mathbf{y(k)} = \begin{bmatrix} d(k)-\overline{d} & u(k)-\overline{u}\end{bmatrix}^T$ and $\mathbf{v(k)} = \begin{bmatrix} b(k)-\overline{b} & s(k)-\overline{s}\end{bmatrix}^{\mathbf{T}}$, the input and output vector, respectively. n_a and n_b are the orders of the control outputs and control inputs. $\mathbf{A_i}$ and $\mathbf{B_i}$ are model parameters whose values need to be determined by system identification.

To have an accurate model, we need to identify the operating points of the controlled system. For regulatory control, the operating points of the outputs (*i.e.*, $\overline{d}$ and $\overline{u}$) are typically chosen to lie close to the reference values (*i.e.*, R_d and R_u) [24]. Meanwhile, the operating points of the inputs are identified (*i.e.*, $\overline{b}$ and $\overline{s}$) by preliminary experiments such that the operating points of the outputs can be reached [23].

For system identification, we need to first determine the right orders for the system, *i.e.*, n_a and n_b, in the difference equation (1). The order values are normally a compromise between model simplicity and model accuracy. In this paper, we test different system orders by using pseudo-random digital white noise [23] to stimulate the system and then measure the control outputs (*i.e.*, $d(k)$ and $u(k)$) in each control period. Our experiments are conducted on the testbed introduced in detail in Section V. Based on the collected data, we use the *Least Squares Method (LSM)* to iteratively estimate the values of parameters $\mathbf{A_i}$ and $\mathbf{B_i}$. In this paper, we choose $n_a = 1$ and $n_b = 1$ for a trade-off between model complexity and model accuracy.

We then generate another sequence of white noise to validate the results of system identification. Figure 4 shows the measured outputs from the open-loop system and the predicted outputs from the model. The errors measured in *Root Mean Squared Error (RMSE)* are sufficiently small (*i.e.*, 0.113 and 0.031 for the average matching delay and the CPU utilization, respectively). We can see that the predicted outputs of the selected model are sufficiently close to the actual system outputs. Therefore, the resultant system model from our system identification is:

$$\mathbf{y(k)} = \mathbf{A_1 y(k-1)} + \mathbf{B_1 v(k-1)}, \tag{2}$$

where A_1 and B_1 are 2×2 constant matrices whose values are determined by system identification.

C. Controller Design

We apply the *Linear Quadratic Regulator (LQR)* control theory to design the controller based on the system model (2). LQR is an optimal control technique that can deal with coupled MIMO control problems and has small computational overhead at runtime. To design the controller, we first convert our system model to a state space model. The state variables are defined as follows:

$$\mathbf{x(k)} = \begin{bmatrix} \mathbf{e}(k) \\ \mathbf{e_I}(k) \end{bmatrix}, \tag{3}$$

where $\mathbf{e}(k) = \mathbf{R_{ef}} - \begin{bmatrix} d(k) & u(k) \end{bmatrix}^T$ and $\mathbf{e_I}(k) = \mathbf{e_I}(k-1) + \mathbf{e}(k)$ are the control error vector and the accumulated control error vector, respectively. Our final state space model of the controlled system is:

$$\begin{bmatrix} \mathbf{e}(k+1) \\ \mathbf{e_I}(k+1) \end{bmatrix} = \mathbf{A} \begin{bmatrix} \mathbf{e}(k) \\ \mathbf{e_I}(k) \end{bmatrix} + \mathbf{B}\mathbf{v}(k) + \begin{bmatrix} \mathbf{I}-\mathbf{A_1} \\ \mathbf{0} \end{bmatrix} \mathbf{r}, \tag{4}$$

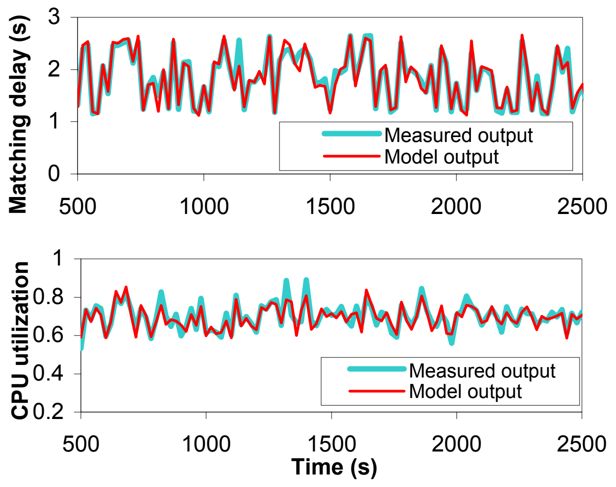


Fig. 4. System model validation

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{A_1} & \mathbf{0} \\ \mathbf{I} & \mathbf{I} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} -\mathbf{B_1} \\ \mathbf{0} \end{bmatrix},$$
$$\mathbf{r} = \mathbf{R_{ef}} - \begin{bmatrix} \bar{d} & \bar{u} \end{bmatrix}^T.$$

Based on the state space model (4), we design the LQR controller by choosing gains to minimize the following quadratic cost function:

$$\mathbf{J} = \sum_{k=1}^{\infty} \begin{bmatrix} \mathbf{e(k)^T} & \mathbf{e_I(k)^T} \end{bmatrix} \mathbf{Q} \begin{bmatrix} \mathbf{e}(k) \\ \mathbf{e_I}(k) \end{bmatrix} + \sum_{k=1}^{\infty} \mathbf{v^T(k)Rv(k)}, \tag{5}$$

where $\mathbf{Q}$ and $\mathbf{R}$ are weighting matrices that determine the trade-off between the control errors and control efforts. The first item in (5) represents the control errors and accumulated control errors. By minimizing the first item, the closed-loop system can converge to the desired set points. The second item in (5) represents the control efforts. Minimizing the second item ensures that the controller will minimize the changes in the control inputs, *i.e.*, the changes of the job budget and batching window size. A general rule to determine the values of $\mathbf{Q}$ and $\mathbf{R}$ is that a larger $\mathbf{Q}$ leads to faster response to workload variations, while a larger $\mathbf{R}$ makes the system less sensitive to system noise. The LQR controller is designed by using the Matlab command *dlqry* to solve the optimization problem (5). The controller gain matrix $\mathbf{K}$ is as follows:

$$\mathbf{K} = \begin{bmatrix} \mathbf{K_P} & \mathbf{K_I} \end{bmatrix}, \tag{6}$$

where $\mathbf{K_P}$, $\mathbf{K_I}$ are constant controller parameters for the error vector $\mathbf{e}(k)$ and the accumulating error vector $\mathbf{e_I}(k)$, respectively, so that the cost function (5) is minimized. Consequently, the control inputs in the k^{th} control period, *i.e.*, the job budget and the batching window size, are computed as:

$$\begin{bmatrix} b(k) - \bar{b} \\ s(k) - \bar{s} \end{bmatrix} = -\mathbf{K} \begin{bmatrix} \mathbf{e}(k) \\ \mathbf{e_I}(k) \end{bmatrix} \tag{7}$$

D. Control Analysis for Model Variations

A fundamental benefit of the control-theoretic approach is that it gives us theoretical confidence in system stability, even when the controller is used under different working conditions. In this subsection, we analyze the system stability when the integrated controller is applied to a system whose model is different from the nominal model described by (2). One of the major model variations is due to the varying arrival time of updates within the batching window. As an example, we give the stability analysis for varying arrival time of updates.

Since most updates arrive within a certain batching window, we have $w(k) \leq s(k)$. Therefore, we assume $w(k) = g \cdot s(k)$, where $0 \leq g \leq 1$. We define g as the *waiting-time factor*. In system identification, we assume that the average waiting time of updates is half of the batching window size, *i.e.*, $g = 0.5$. However, in real systems, the average waiting time of updates is unknown a priori. To verify that the closed-loop system is stable when the average waiting time is any proportion of the batching window size, here we outline the general steps to analyze the stability:

1) We model the variation of the average arrival time as g in the matrix B_1 in the system model (2) as follows:

$$\mathbf{B_1'} = \begin{bmatrix} b_{11} & b_{12} - 0.5 + g \\ b_{21} & b_{22} \end{bmatrix}, \tag{8}$$

where $\mathbf{B_1'}$ is the varied matrix of $\mathbf{B_1}$ at runtime. b_{12} models the relationship between the window size and matching delay;

2) Derive the closed-loop system model by substituting the derived control inputs $\mathbf{v}(k)$ into the system model (2) by replacing $\mathbf{B_1}$ with $\mathbf{B_1'}$. The closed-loop system model is in the following form:

$$\begin{bmatrix} \mathbf{e}(k) \\ \mathbf{e_I}(k) \end{bmatrix} = (\mathbf{A} - \mathbf{B'K}) \begin{bmatrix} \mathbf{e}(k-1) \\ \mathbf{e_I}(k-1) \end{bmatrix} + \begin{bmatrix} \mathbf{I} - \mathbf{A_1} \\ \mathbf{0} \end{bmatrix} \mathbf{r}, \tag{9}$$

where $\mathbf{B'} = \begin{bmatrix} -\mathbf{B_1'} & \mathbf{0} \end{bmatrix}$, and $\mathbf{r} = \mathbf{R_{ef}} - \begin{bmatrix} \bar{d} & \bar{u} \end{bmatrix}^T$.

3) Derive the stability condition of the closed-loop system described by (9). According to control theory, the closed-loop system is stable if all eigenvalues of the matrix $(\mathbf{A} - \mathbf{B'K})$ are located inside the unit circle.

Following the steps above, we have proven that the closed-loop system is stable when g varies from 0 to 1 at runtime, which means that the system can be guaranteed to be stable no matter when the updates arrive. Similarly, we can also analyze the stability with other model variations by following the steps above.

V. System Implementation

In this section, we introduce the testbed and implementation details of the control loops.

A. Testbed

Our testbed includes two servers connected via a network switch. Server 1 is equipped with Intel Core 2 Duo Xeon 5160 3.0GHz and Server 2 is equipped with AMD Athlon 64x2 4200+ 1.0GHz. Both of them run openSUSE 10.3 (2.6.22) and JDK 1.6.0. We use the INFOD system introduced in Section I as a representative information dissemination system to test our control loops. The INFOD registry is implemented in Oracle Database 11.1g on Server 1 to run the metadata matching processes. Since our objective is to evaluate the performance of the control loops running on Server 1, all INFOD publishers, consumers and subscribers are implemented on Server 2 to simplify the experimental setup.

According to queuing theory, the arrival of requests at a server can often be realistically characterized as a Poisson process [25]. Without loss of generality, we use updates whose arrival pattern follows the Poisson process with specific values of λ (*i.e.*, the average number of arriving updates every second) in this paper. We refer to λ in Poisson process as the *update arrival rate*. The updates in our testbed have some typical arrival rates, ranging from 2 to 10. Note that our control algorithm is not limited to Poisson process. As proven in Section IV-D, our control algorithm is insensitive to arrival patterns of updates.

The process of metadata matching is implemented as an Oracle PL/SQL procedure and configured as an event-based task with a priority of 3 out of the five priorities (1 to 5, with 1 as the highest) provided by the Oracle database scheduler to let the controllers have the highest priority to run.

B. Integrated Control Loop

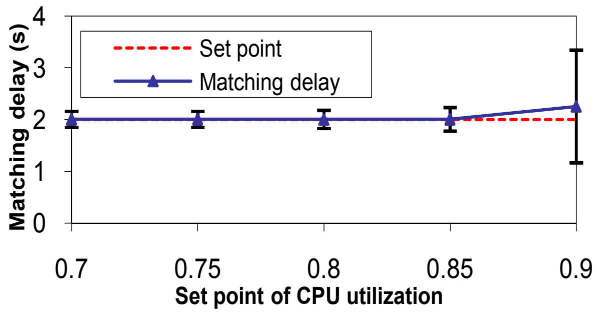
We now introduce the implementation details of each component in the integrated control loop.

Updates Batcher: The updates batcher is implemented as a two-threaded daemon in Java. One thread listens on a TCP/IP socket to receive updates from the clients and records the arrival time of each update. The other thread serves as one of the actuators in the integrated control loop and is invoked from time to time based on the batching window size calculated by the integrated controller. Upon each invocation, it calculates the average waiting time of each update within the batching window and invokes the scheduler in the registry. The updates batcher communicates with the registry through JDBC (Java DataBase Connectivity), a Java package enabling Java programs to execute SQL statements in databases [26].

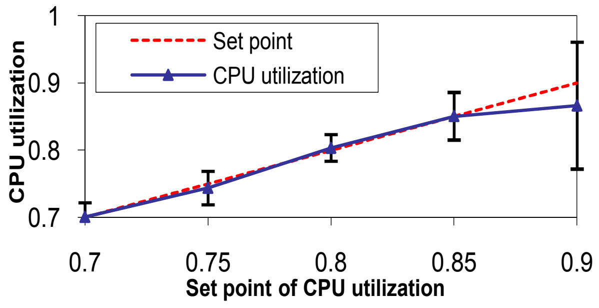
Scheduler: The scheduler is an application-level Oracle PL/SQL procedure implemented on top of the internal scheduler of the Oracle database. Oracle provides a package called DBMS_SCHEDULER from which we can call a collection of scheduling functions and procedures to manage jobs (*e.g.*, create, configure and schedule) [27]. The application-level scheduler is invoked by the updates batcher and serves as the other actuator for controlling the average matching delay and CPU utilization.

Delay and CPU utilization Monitor: The monitor is implemented as a PL/SQL procedure. The monitor calculates the average matching delay as the sum of the average waiting time of each update and average processing time of all high-priority subscriptions reevaluated in this control period, and the CPU utilization in this control period based on the statistical information in the view of V$OSSTAT in the database [28].

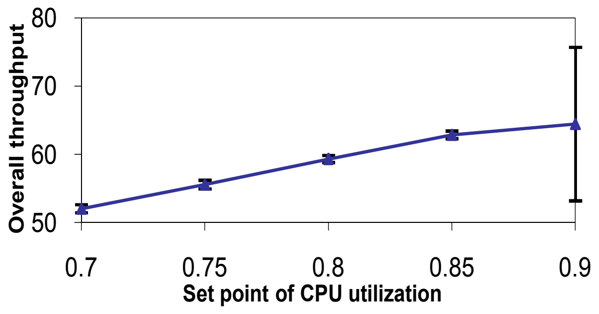
Integrated Controller: The integrated controller is a periodic job in the database. The controller is assigned the highest priority (*i.e.*, 1) such that it can run periodically even when the system becomes overloaded. The period of the integrated controller is selected based on a trade-off between the sensitivity to system noise and reaction speed of the controller. On one hand, each control period should be long enough to include multiple batching windows such that the influence of the system disturbance and noise incurred to the controlled variables can be reduced. On the other hand, a longer control period leads to slower reaction to workload variations. In this paper, the batching window size calculated by the integrated controller is within the range of $(1s, 4s)$ (*i.e.*, the operating region) based on our preliminary experiments. Therefore, the control period is set as $20s$ to include at least 5 batching windows. The measured average overhead of running the whole control loop is about $100ms$, roughly 0.5% of a control period. This amount of overhead should be accpetable to most systems.



(a) Matching delay



(b) CPU utilization

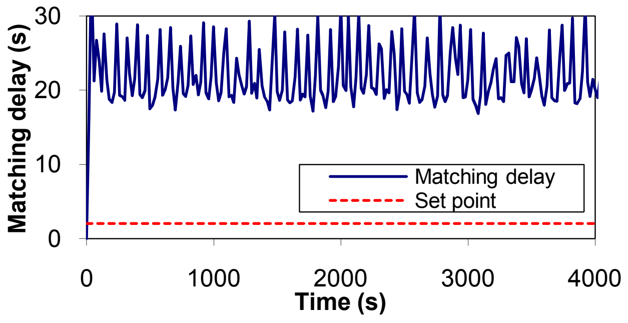


(c) Overall throughput

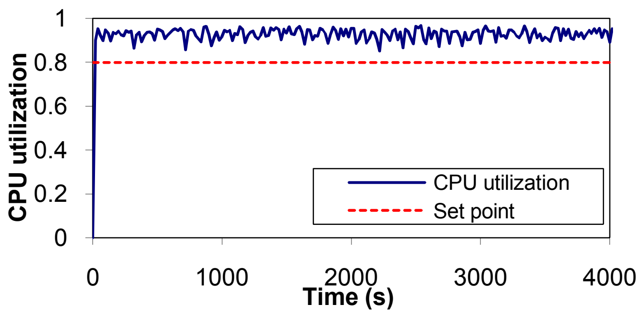
Fig. 5. Selection of set points for CPU utilization control

VI. Experimentation

In this section, we first test different set points of CPU utilization to justify the selection of the set point for CPU utilization employed in our experiments. We then evaluate the performance of the integrated controller. Finally, we compare it with two baselines.

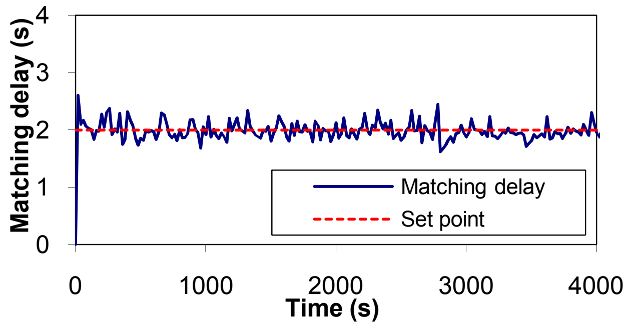


(a) Matching delay

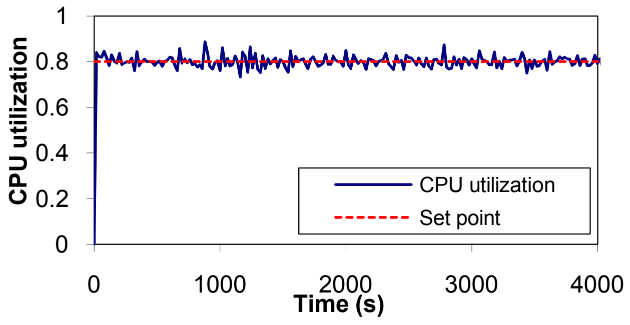


(b) CPU utilization

Fig. 6. A typical run of the open-loop system



(a) Matching delay



(b) CPU utilization

Fig. 7. A typical run of the integrated controller

In all of the experiments, we use $2s$ as the set point of the average matching delay and 0.8 as the set point of the CPU utilization. The steady states of both the average matching delay and CPU utilization are defined as $\pm 10\%$ of the set point.

A. Set Point Selection for CPU Utilization Control

As explained in Section IV-A, the selection of the set point of CPU utilization is a trade-off between the control accuracy of the matching delay and overall throughput. In this subsection, we test the effect of set points of CPU utilization on the control accuracy and overall throughput (*i.e.*, the number of reevaluated subscriptions per second) with experiments.

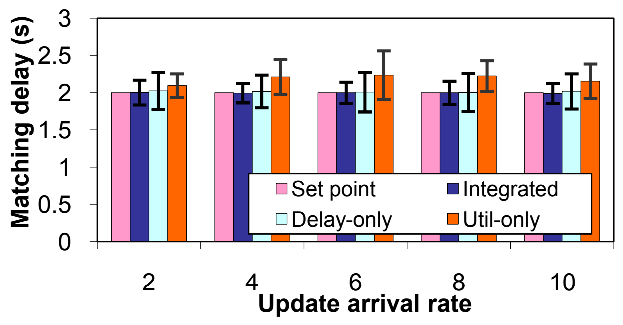
We first run the integrated controller with different set points of CPU utilization, from 0.70, 0.75, 0.80, 0.85 to 0.90, while keeping the set point of the average matching delay constantly at $2s$. The reason why we do not vary the set point of the matching delay is that the constraint of matching delay is usually specified by customers. We then plot the mean and standard deviation of 50 outputs in the steady state for the average matching delay, CPU utilization, and the overall throughput in Figure 5. As shown in Figures 5(a) and (b), both the average matching delay and the CPU utilization have mean values approximately equal to their set points with only small deviations when the set point of the CPU utilization is smaller than 0.9. However, as the set point increases to 0.9, both the average matching delay and CPU utilization have mean values deviating from the set point with significant oscillations. This is because the system becomes overloaded and the system model is nonlinear when the CPU utilization approaches 0.9. As shown in Figure 5(c), the overall throughput steadily improves as the set point increases from 0.7 to 0.85 and reaches the maximal value at the CPU utilization of 0.85 with small oscillations. However, as the set point increases to 0.90, the overall throughput begins to oscillate significantly due to the large oscillation of the CPU utilization.

To ensure that the controller works safely in the operating region, we choose 0.80 as the set point to allow some leeway for the nonlinear region. This experiment gives us a good reference to choose the set point for CPU utilization.

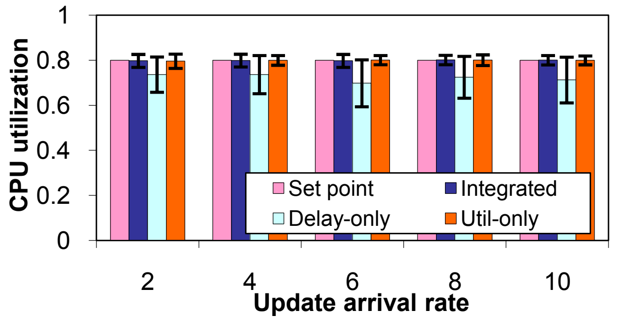
B. Control Performance of the Integrated Controller

In this subsection, we first demonstrate the performance of the open-loop system to show the importance of controlling the matching delay and CPU utilization. We then evaluate the performance of the integrated controller with different update arrival rates.

The open-loop system is the system without any matching delay or CPU utilization management, in which all the subscriptions are reevaluated upon the arrival of each update. In this experiment, we use an update arrival rate of 4 (*i.e.*, $\lambda = 4$), which is a typical arrival rate in real systems. Figure 6 shows the average matching delay and CPU utilization of the open-loop system with 400 subscriptions. We can see that reevaluating all subscriptions upon each update causes severe system overload (*i.e.*, the CPU utilization is almost 0.93 (there is some CPU time for I/O waiting.)) and unacceptably long delays (*i.e.*, the delay is greater than $30s$ at times). This experiment shows that unbounded matching delay and system

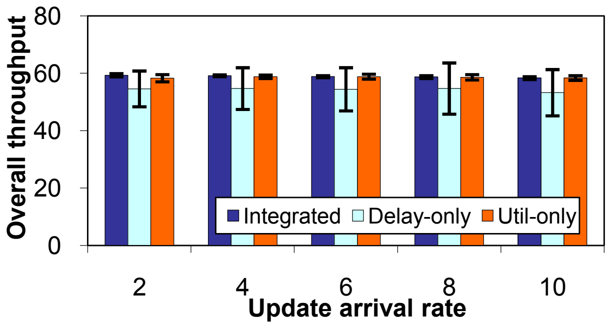


(a) Matching delay

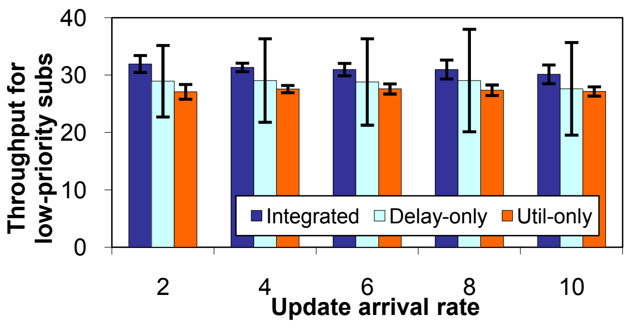


(b) CPU utilization

Fig. 8. Control accuracy comparison among Delay-only, Util-only, and the proposed integrated controller



(a) Overall throughput



(b) Throughput for low-priority subscriptions

Fig. 9. Throughput comparison among Delay-only, Util-only, and the proposed integrated controller

overutilization will occur without matching delay or CPU utilization management in information dissemination systems. Figure 7 shows that the integrated controller successfully achieves the desired matching delay and CPU utilization with only small oscillation.

In real information dissemination systems, the update arrival rate may vary due to different factors, such as working hours. To evaluate the integrated controller under varying arrival rates, we run experiments and plot the mean and standard deviation of 50 outputs in the steady state when the arrival rates vary from 2, 4, 6, 8 to 10. The standard deviation indicates the intensity of oscillation of the outputs in the steady state. From Figure 8, we can see that the mean of the average matching delay and CPU utilization are all around the set points with only small standard deviations (the largest is $0.18s$ and 0.03, respectively). This experiment shows that the integrated controller can, precisely, achieve the desired average matching delay and CPU utilization under different update arrival rates.

C. Comparison with Baselines

One of the advantages of the integrated controller is that it can simultaneously control both the matching delay and CPU utilization to achieve maximized system throughputs so that more valuable information can be timely disseminated. To highlight this advantage, we compare our integrated controller with two baselines: *Delay-only* and *Util-only*, in terms of control accuracy, overall throughput, and throughput for low-priority subscriptions.

Comparison with Delay-only: Delay-only is a SISO controller that is proposed in [7]. It has a fixed batching window size and controls only the average matching delay of the high-priority subscriptions by manipulating the job budget. We tune it with different batching window sizes to make it have the best overall throughput while controlling the average matching delay with an accuracy close to that of the integrated controller. We find that a window size of $1.95s$ produces the best results. We test Delay-only with different update arrival rates and plot the mean and standard deviation of 50 outputs in the steady state for average matching delay, CPU utilization, overall throughput, and throughput for low-priority subscriptions. As shown in Figures 8(a), the average matching delay of Delay-only with different arrival rates is all approximately equal to the set point with small oscillations (with the largest standard deviation as $0.25s$.). However, as shown in Figure 8(b), Delay-only has a poor CPU utilization because it only controls matching delay. As a result, Delay-only has a poor overall throughput and a poor throughput for low-priority subscriptions, as shown in Figures 9(a) and (b).

Comparison with Util-only: Util-only is another SISO controller that only controls CPU utilization by manipulating the job budget with a fixed batching window size. We first fix the batching window size for Util-only at the mean of the batching window size of the integrated controller in the steady state (*i.e.*, $1.8s$). We plot the mean and standard deviation of 50 outputs in the steady state for average matching delay and CPU utilization under different update arrival rates, as shown in Figures 8. We can see that Util-only violates the specified delay constraint though its CPU utilization converges to the set point with small oscillations. To have a fair comparison, we then tune Util-only with different batching window sizes so that it has an average matching delay close to the set point of $2s$. By iterative tuning and testing, we find that a batching window size of $1.6s$ gives us the best results. We then compare Util-only with the integrated controller in terms of overall throughput and throughput for low-priority subscriptions. As shown in Figures 9(a), since both Util-only and the integrated controller control the CPU utilization to the set point of 0.8, they have almost the same overall throughput. However, due to a smaller batching window size, Util-only has a smaller job budget, which results in a lower throughput for low-priority subscriptions, as shown in Figure 9(b).

The two experiments show that only controlling either matching delay or CPU utilization results in poor system throughputs or violates the specified delay requirement. In contrast, the integrated controller, by controlling both matching delay and CPU utilization, can meet the specified constraint for matching delay and achieve maximized possible system throughputs.

VII. Conclusions

Existing information dissemination systems control metadata matching delay and resource utilization separately. However, both bounded matching delay and maximized system throughputs are critical for information dissemination systems to find the maximum number of matched results and then disseminate valuable information within application-specified time constraints. In this paper, we have proposed an integrated controller to control both the matching delay of all the high-priority subscriptions and the CPU utilization of the registry server in an example information dissemination system. Our empirical results on a physical testbed demonstrate that our solution can guarantee the timeliness requirements while achieving maximized system throughputs.

Acknowledgements

This work was supported, in part, by NSF under a CAREER Award CNS-0845390 and a CSR grant CNS-0720663, by ONR under grant N00014-09-1-0750, and 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. We would also like to thank the anonymous reviewers for their valuable comments.

References

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

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

[3] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu, "Dynamic voltage scaling in multi-tier web servers with end-to-end delay control," *IEEE Transactions on Computers*, vol. 56, no. 4, pp. 444–458, 2007.

[4] Y. Wang, X. Wang, M. Chen, and X. Zhu, "Power-efficient response time guarantees for virtualized enterprise servers," in *IEEE Real-Time Systems Symposium (RTSS)*, Dec. 2008.

[5] M. Chen, C. Nolan, X. Wang, S. Adhikari, F. Li, and H. Qi, "Hierarchical utilization control for real-time and resilient power grid," in *Euromicro Conference on Real-Time Systems (ECRTS)*, Jul 2009.

[6] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury, "Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache web server," in *Network Operations and Management Symposium*, 2002.

[7] M. Chen, X. Wang, R. Gunasekaran, H. Qi, and M. Shankar, "Control-based real-time metadata matching for information dissemination," in *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug 2008.

[8] D. Kostić, A. C. Snoeren, A. Vahdat, R. Braud, C. Killian, J. W. Anderson, J. Albrecht, A. Rodriguez, and E. Vandekieft, "High-bandwidth data dissemination for large-scale distributed systems," *ACM Transactions on Computer Systems*, vol. 26, no. 1, pp. 1–61, 2008.

[9] S. Voulgaris, E. Riviére, A.-M. Kermarrec, and M. van Steen, "Sub-2-sub: Self-organizing content-based publish subscribe for dynamic large scale collaborative networks," in *International Workshop on Peer-To-Peer Systems (IPTPS)*, Feb. 2006.

[10] A. Iamnitchi and I. Foster, "Interest-aware information dissemination in small-world communities," in *High Performance Distributed Computing (HPDC)*, 2005.

[11] G. S. Choi, J.-H. Kim, D. Ersoz, A. B. Yoo, and C. R. Das, "Coscheduling in clusters: Is it a viable alternative?" in *Super Computing*, 2004.

[12] E. Frachtenberg, D. G. Feitelson, F. Petrini, and J. Fernandez, "Adaptive parallel job scheduling with flexible coscheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 11, 2005.

[13] O. Mutlu and T. Moscibroda, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems," in *International Symposium on Computer Architecture (ISCA)*, 2008.

[14] B. Towles and W. J. Dally, "Guaranteed scheduling for switches with configuration overhead," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, 2003.

[15] X. Cao, J. James, J. Li, and C. Xin, "Group schedule serialized traffic in optical burst switching networks," in *International Conference on Broadband Communications, Networks, and Systems (BROADNETS)*, Sept. 2007.

[16] 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. 2000.

[17] T. Gustafsson and J. Hansson, "Dynamic on-demand updating of data in real-time database systems," in *ACM Special Interest Group on Applied Computing (SIGAPP)*, Mar. 2004.

[18] T. Gustafsson and J.Hansson, "Data management in realtime systems: a case of on-demand updates in vehicle control systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, May 2004.

[19] 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. 10, Oct. 2004.

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

[21] 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.

[22] G. F. Franklin, J. D. Powell, and M. Workman, *Digital Control of Dynamic Systems, 3rd edition*. Prentice Hall, 1997.

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

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

[25] U. N. Bhat, *An Introduction to Queueing Theory: Modeling and Analysis in Applications*. Birkhäuser, 2008.

[26] *Oracle Database JDBC Developer's Guide and Reference 11g Release 1 (11.1)*, Oracle, Jul. 2008.

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

[28] *Oracle Database Reference 11g Release 1 (11.1)*, Oracle, Aug. 2008.