Control-Based Quality Adaptation in Data Stream Management Systems

Yi-Cheng Tu[1], Mohamed Hefeeda[2], Yuni Xia[1], Sunil Prabhakar[1], and Song Liu[1]

[1] Purdue University, West Lafayette, IN 47906, U.S.A.
[2] Simon Fraser University, Surrey, BC, Canada

Abstract. Unlike processing snapshot queries in a traditional DBMS, the processing of continuous queries in a data stream management system (DSMS) needs to satisfy quality requirements such as processing delay. When the system is overloaded, quality degrades significantly thus load shedding becomes necessary. Maintaining the quality of queries is a difficult problem because both the processing cost and data arrival rate are highly unpredictable. We propose a quality adaptation framework that adjusts the application behavior based on the current system status. We leverage techniques from the area of control theory in designing the quality adaptation framework. Our simulation results demonstrate the effectiveness of the control-based quality adaptation strategy. Comparing to solutions proposed in previous works, our approach achieves significantly better quality with less waste of resources.

1 Introduction

The past few years have witnessed the emergence of applications such as network monitoring, financial data analysis, location-based services, and sensor networks in which information naturally occurs in the form of continuous data streams. These applications do not fit in the data model and querying paradigm of traditional Database Management Systems (DBMSs). Data stream management systems (DSMSs) such as Aurora [1], STREAM [2], and TelegraphCQ [3] are built for data management and query processing in the presence of multiple, continuous, and time-varying data streams. Instead of being stored in advance on disks, streaming data elements arrive on-line and stay only for a limited time period in memory. Consequently, a DSMS has to handle the data elements before the buffer is overwritten by new incoming data elements.

Processing of data streams brings great challenges to DBMS design for two major reasons. First, stream data are continuously generated in large volumes by external sources such as a sensor network. Second, the data are often collected from remote sites and are prone to loss and delay in most cases. From the applications point of view, queries to data streams are also different from those against traditional data. While snapshot queries (e.g. *show me the current room temperature*) are mainstream in traditional DBMSs, persistent queries that output results periodically (e.g. *show me the temperature readings every 5 minutes*) are

K.V. Andersen, J. Debenham, and R. Wagner (Eds.): DEXA 2005, LNCS 3588, pp. 746–755, 2005.

very common in data stream applications. Unlike those in a traditional DBMS, queries in a DSMS can be processed with various levels of timeliness, data reliability, and precision [4,5]. For example, the above query can be rewritten as *every 5 minutes return the temperature readings that are collected no longer than 5 seconds ago*. Here the extra parameter of "5 seconds" represents a requirement on data freshness. We call such parameters the *quality* of the query, much like QoS parameters in multimedia applications. Quality support for queries is a critical issue in DSMS design as it is directly related to user satisfaction.

Quality guarantees for multiple streams/queries in a DSMS are difficult to maintain due to limitations of the physical resources and fluctuations in the application's behavior. A data stream system could contain a large number of streams and continuous queries. Quality will degrade when the system is overloaded. Even with careful admission control, fluctuations of resource availability (e.g. bandwidth variations of a network link) and application resource usage (e.g. bursty traffic) may still cause temporary overloading that interferes with the proper processing of queries. To deal with such variations, the DSMS needs to automatically adjust the quality of individual queries (*adaptation*) at runtime. For example, in case of overloading, data tuples can be discarded to save costs for query processing (*load shedding*). In considering quality adaptations, we always have to answer the following two questions:

1. *When to perform adaptation?*
2. *What is the ideal magnitude of adaptation?*

In this paper, we propose an approach that answers these two questions.

Streaming data are intrinsically dynamic with respect to the arrival patterns [6]. As an example, Figure 1 shows the traces of real TCP traffic recorded from a cluster of web servers. Note the number of packet arrivals per unit time fluctuates within the range of $[120, 450]$ and no obvious period can be observed. Furthermore, the processing costs of data tuples are also difficult to predict. As a result, estimation of instantaneous and future resource consumption could easily fail. Thus, reservation-based QoS-provisioning strategies [7] may not be applicable to DSMSs. This opens great opportunities for optimization towards an auto-configuring DSMS that smartly adjusts quality levels of streams in response to fluctuations of system status and input rates.

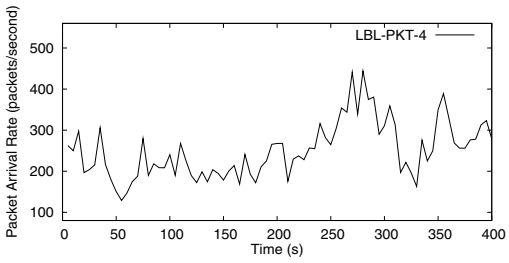

Fig. 1. Fluctuations in the arrival rate of real TCP traffic

In this paper, we propose a solution that is derived from classical control theory to address the above challenge. We formulate quality adaptation in DSMS as

a feedback control problem. Despite the difficulties of modeling the DSMS behavior, we utilize well-established tools provided by control theory to estimate the model and develop formulas to make adaptation decisions based on the derived model. The unpredictable traffic pattern and processing cost are modeled as disturbances to the control loop. We present experimental results that demonstrate the validity of our method. To the best of our knowledge, this is the first work that applies control theory to data stream management research. Our ongoing research shows that the control-based approach may also be useful in a broader range of topics in database research.

2 Related Work and Our Contribution

Current works in DSMS study strategies to maximize quality parameters such as query precision [8] and loss tolerance [1]. Little attention has been paid to quality adaptation. Research that is most closely related to our work is QoS-aware load shedding introduced in the context of the Aurora project [5]. In Aurora, a load shedding roadmap (LSRM) is constructed to determine the target and amount for adaptation. LSRM generates a list of possible shedding plans and sorts them by their expected returns in CPU cost. The amount of shedding is given by a fixed percentage P that is obtained empirically. At runtime, the amount of load shedding increases by P as long as there is overloading. In [9], load shedding strategy that minimizes the loss of accuracy of aggregation queries in DSMS is discussed. The use of feedback control is inspired by the work of Lu *et al* [10], which deals with the problem of real-time scheduling.

The major contribution of this work lies in its improvement on current research on load shedding [5,9] in the following aspects: (i) In previous works, adaptation (shedding) decisions depend heavily on steady-state estimates of data traffic and processing costs. The fluctuations in data arrival are regarded negligible. Therefore, adaptation does not work well on fluctuating/bursty data inputs. Our control-based approach explicitly takes these uncertainties into account. (ii) The magnitude of adaptation (e.g., percentage of load to shed) is determined using simple rules of thumb. As a result, adaptation is often imprecise: its magnitude can either be insufficient (undershoot) to achieve desirable quality or over-killing (overshoot) that leads to waste of resources. Furthermore, system tuning is tedious due to the use of rules of thumb. Our solution allows us to analytically tune the controller to achieve guaranteed performance.

3 The DSMS Model

We follow a push-based data stream query processing model (Fig 2), which is a generalization of those of STREAM [2] and Aurora [1]. Streams (S1 – S3) are generated in remotely-located devices and sent to the DSMS via a network. Upon arriving at the DSMS, each data tuple from a stream is processed by several database *operators* chained as a pipeline. The pipelines are called *query plans*, which are formed by the DSMS for active queries. For example, query

Q1 involves a join operation (O1) between S1 and S2, followed by a selection operation (O3). Execution of operators are controlled by a resource *scheduler*. In this model, we do not dictate a specific policy for the scheduler. Our quality adaptation framework works for arbitrary scheduling policies. Before being selected for processing, data tuples are kept in waiting queues (q1 – q4) for the operators. Various system resources including CPU [5], memory [11], and network bandwidth [12] could be the bottleneck in DSMS query processing under different situations. In this paper, we concentrate on CPU usage. Each operator O_i has an estimated CPU cost denoted as c_i.

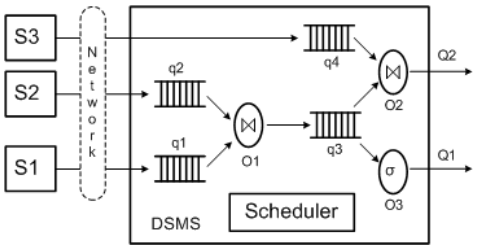

Fig. 2. System model of a typical DSMS

Depending on the applications we are interested in, the selection of quality parameters varies. In this research, we concentrate on *tuple delays* (i.e. time elapsed between the generation and the end of processing of a tuple) as we believe this is the most critical quality for many applications. Specifically, we allow users to choose their delay requirements from a set of discrete values. A violation of tuple delay requirement is called *deadline miss*. We control deadline misses by regulating the system load (*load shedding*). A trivial strategy to maintain low deadline miss ratio would be shedding a large fraction of the load at all times. However, high data loss due to load shedding is also undesirable therefore the problem becomes how to control deadline misses with less data loss.

4 The Control-Based Quality Adaptation Framework

Feedback control refers to the operation of manipulating system behavior by adjusting system input based on system output (feedback). A well-known example is the cruise control of automobiles. The central idea of feedback control is the feedback control loop, which consists of the following components (Fig 3): (i) *Plant*, which is the system to be controlled; (ii) *Monitor*, which periodically measures the status of the plant; (iii) *Controller*, which compares the current status of the plant sensed by the monitor versus the desired status that is set beforehand. The difference between the current status and the desired status is called *error*. The function of the controller is to map the error to an appropriate control signal that is sent to the actuator; (iv) *Actuator*, which adjusts the behavior of the plant in proportion to the control signal.

Control theory provides tools (e.g., Root locus) to efficiently design mappings from the error to the control signal. Furthermore, under the effects of environmental fluctuations, control theory allows us to choose and tune the controller

parameters in order to: (i) guarantee system stability, (ii) provide theoretical bounds on the system outputs, (iii) eliminate or bound the error in the steady state, and (iv) adjust the settling time, which is the time needed for the system to reach the steady state. These features distinguish the feedback control approach from *ad hoc* solutions that use rules of thumb. Readers interested in more details on control theory are referred to [13].

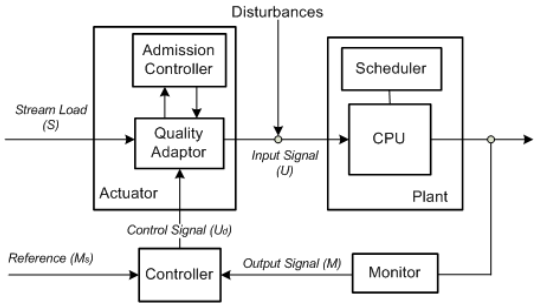

Fig. 3. The control-based quality adaptation framework

4.1 The Feedback Control Loop

Figure 3 also shows the architecture of the proposed control-based quality adaptation framework. The plant is the DSMS, a central part of which is the CPU and its scheduler. Plant status is monitored periodically and output signals are sent to the controller. The monitor is a simple component in this case since the output signal is a variable maintained by the scheduler. We use the Deadline Miss Ratio (M), which is the fraction of tuples that miss their processing deadlines, as the output signal. The fluctuations of data arrival rate and the inaccuracy in CPU cost estimation of operators are modeled as disturbances to the system.

The actuator is divided into two parts: the *Admission Controller* and the *Quality Adaptor*. The former decides whether new streams can be accommodated by the system while the latter adjusts each stream to control the load on the CPU. We use a unitless quantity, CPU utilization (U), to represent the load on CPU. U is defined as the total CPU time needed to process all the data tuples that arrive within one unit time. Note U is the input signal to the plant. The quality adaptor is implemented as a load shedder, which drops tuples from the admitted streams to reduce the load on the CPU. The percentage of tuples to drop (i.e., the shed factor) is determined by the controller, while choosing the victim tuples to drop depends on the implementation of the load shedder itself. We adopt the simplest load shedding strategy: choosing victim tuples randomly. This strategy is not optimized for global utility as streams and queries may have different priorities. A strategy that takes these differences into account is called semantic adaptation [5]. We plan to investigate the complications of adopting semantic adaptation in our control-based paradigm in the future.

The feedback control loop works as follows: a desired deadline miss ratio M_s is specified by the system administrator. The monitor measures the output signal M at the end of every sampling period. The controller compares M to M_s and generates the control signal: the amount of load that needs to be shed. For

convenience, we define the control signal (U_d) to be the percentage of load that needs to be kept. In other words, $1 - U_d$ is the shedding factor. The controller then sends the load change U_d to the actuator. Clearly, the generation of input signal U_d is the key point in this loop and will be discussed in Section 4.2. Intuitively, U and M are positively related: when U increases, the load put on the CPU also increases therefore more deadline misses occur.

4.2 Controller Design

Now we describe how the input signal is generated by the controller. First we denote $M(k)$ and $U(k)$ as the output and input of the plant at the k-th sampling period. We start the controller design by constructing a dynamic model for the plant. Unlike mechanical systems, our DSMS is not likely to be modeled using known laws of nature. Fortunately, modern control theory provides *system identification* techniques [14] to estimate the model of complex systems. The idea is to model the plant as a difference equation with unknown parameters. For example, the DSMS could be modeled by the following difference equation:

$$M(k) = a_1 M(k-1) + a_2 M(k-2) + b_1 U(k-1) + b_2 U(k-2) \tag{1}$$

where a_1, a_2, b_1, b_2 are system-specific parameters. Then we can determine the order and parameters of the above difference equation experimentally. Specifically, we subject the DSMS of interest with synthetic traffic loads (with white noise) and measure the output. By doing this for a number of iterations we obtain the parameters using Least Square Estimation [14]. The open loop model shown in Eq (1) can be converted into a transfer function in z-domain:

$$G(z) = \frac{M(z)}{U(z)} = \frac{b_1 z + b_2}{z^2 - a_1 z - a_2}. \tag{2}$$

We use a standard Proportional-Integral (PI) [13] controller to compute the input signal as the following:

$$U(k) = U(k-1) + g\big((M_s - M(k)) - r(M_s - M(k-1))\big) \tag{3}$$

where g, r are controller constants. The transfer function of the PI controller is $C(z) = \frac{g(z-r)}{z-1}$. To finish the controller design, we have the following closed loop transfer function:

$$T(z) = \frac{C(z)G(z)}{1 + C(z)G(z)}. \tag{4}$$

As mentioned earlier, the major advantage of the control-based approach over the rules-of-thumb method is that we can analytically tune the controller to achieve guaranteed performance in terms of stability, steady state error, maximum overshoot, and convergence time. In our case, we can use well-known techniques such as Root Locus (available in MATLAB) to tune controller parameters (g and r) based on Eq (4).

Since the quantity $U(k)$ obtained from Eq (3) is the desirable load injected into the plant, we have $U_d(k) = U(k)/S(k)$ where $S(k)$ is the real load generated

by the streams at period k. Unfortunately, $S(k)$ is unknown when we calculate $U_d(k)$. We could use $S(k-1)$ to estimate $S(k)$.[1]

Setting sampling period. The length of sampling period has profound effects on controller design and is a compromise between performance and cost. A general rule is to set the sampling frequency to be as least twice of the signal frequency [13], which is the frequency of fluctuations in streams in our system.

4.3 Admission Controller

To avoid the situation of running a large number of queries all under high data loss ratio, we use an admission controller (AC) to help protect the system from unexpected high loads. The AC checks if a newly registered stream can be served by the system given the current load. For a new stream T_i, the theoretical admission condition is:

$$U + h_i r_i \leq U_s \tag{5}$$

where h_i and r_i are the per-tuple cost and arrival rate of T_i, respectively. The target utilization U_s is generated empirically based on M_s [10]. The practical value of the above admission condition is limited for two reasons: (i) U only reflects the current load situation of the system; (ii) both h_i and r_i are unknown when the stream first registers. We remedy the above conditions as follows: we first make estimations of the the requested utilization by taking a moving average of current and historical data. For example, at the k-th control period, an estimation (U_e) of U can be written as

$$U_e(k) = \gamma U(k) + (1-\gamma)\frac{1}{a}\sum_{i=k-a}^{k-1} U(i),\ (0 < \gamma < 1)$$

Then we can use an optimistic admission controller such as $U_e(k) < U_s$. In applying this optimistic condition, we assume the cost of one single stream is small compared to the total CPU capacity. Even if we make a bad decision at the AC, the control loop will provide a second chance to correct this decision.

5 Experimental Results

To validate our idea of using feedback control for quality adaptation in DSMSs, we implement a control loop as shown in Fig 3 on a data stream simulator. The core of the simulator is a CPU scheduler based on the Earliest Deadline First (EDF) policy. One of the nice features of this policy is that it has a *utilization bound* of 1.0 for both periodic and sporadic tasks [15], meaning there will be (asymptotically) zero deadline misses if utilization (U) is lower than 1.0. In our simulator, we set the target deadline miss ratio M_s to 0, this is the same as to set a target utilization U_s (the same U_s in Eq (5)) to anything smaller than 1.0. We set U_s to be 0.9 in our experiments.

[1] We can show by analysis that this has very little impact on the performance of the controller. We have to skip the analysis due to space limitations.

We define four classes of processing delays: 250ms, 500ms, 1s, and 2s. The monitoring cycle is 5 seconds if not otherwise specified. System identification and analytical tuning of the PI controller (details skipped) suggest we set the g and r values to 0.5 and 0.3, respectively. We test our quality framework with both synthetic and real-world stream data. For the synthetic data, we generate streams with update arrival times following the exponential distribution and the b-model [6]. For real-world data, we use the *LBL-PKT-4* dataset from the Internet Traffic Archive (http://ita.ee.lbl.gov/index.html) that contains traffic traces extracted from web servers. The queries to the stream data are simulated in a manner that is similar to Babcock *et al.* [11].

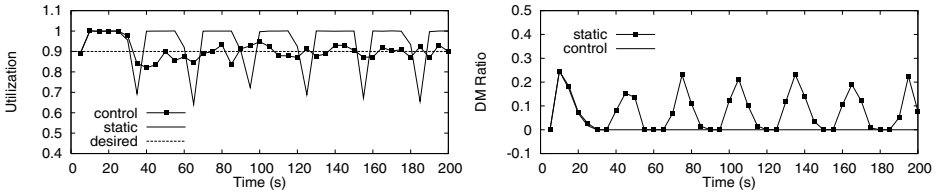

Fig. 4. Utilization and deadline misses by different load shedding strategies

We compare the performance of our control-based method with that of a *static load shedding* strategy similar to the approach in Aurora. The latter uses a rule of thumb in determining U_d: its shed factor is updated in increments of a *base factor* (0.1 unless specified otherwise). Figure 4 shows the results of both shedding methods using the same set of data streams generated with Poisson arrival pattern. The total load of the streams is about 1.4, simulating a constant overloading situation. The control-based strategy converges to a stable state at about 35 seconds and the errors afterwards are kept within ±10% of the desired value. On the contrary, the static shedding strategy shows a zigzag pattern in the achieved utilization with an error range significantly larger than that of the control-based method. As a result, its deadline miss ratio reaches about 20% periodically while there are almost no deadline misses in the control-based experiment.

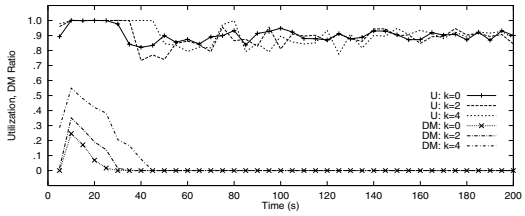

Fig. 5. Performance of control-based load shedding strategy under different precisions of cost estimation

We also investigate the response of our adaptation scheme to imprecise per-tuple cost estimations (Figure 5). We introduce uncertainties to the costs by selecting the real costs of each query iteration randomly from a range around the profiled value. For example, the '$k = 2$' treatment in Figure 5 chooses a cost value for each tuple from the range $[0.5h, 1.5h]$ where h is the profiled CPU

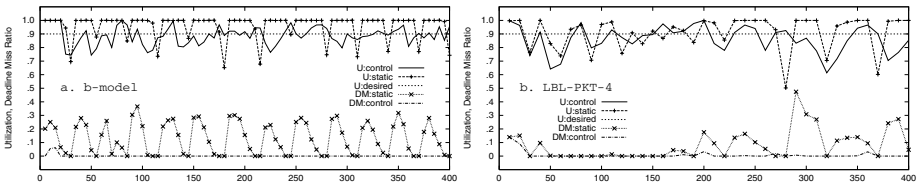

Fig. 6. Performance of different load shedding strategies under different bursty traffic patterns. a. Synthetic *b*-model traffic; b. real TCP traffic.

cost. This range for the case of '$k = 4$' is set to be wider ($[0.1h, 4.1h]$) and the profiled cost h is not even the mathematical expectation of all the possible values. Our quality adaptation strategy handles the incorrect cost estimations well. Comparing to the original experiment where all costs are precisely profiled ('$k = 0$'), the system achieves the same level of stability. The only difference is that the time used to converge to the stable status is longer as uncertainty on costs increases. In other words, the system 'learns' the real value of costs gradually. For all three cases, deadline miss rates are almost zero for most of the time although more can be observed before convergence.

To test system performance under the disturbance of bursty data arrivals, we run experiments with the *b*-model data and real TCP trace (Fig 1). The resource utilization data (Fig 6) are obviously less smooth than those in Fig 4 as more undershoots can be observed. The undershoots become more serious as the traffic gets more bursty. The control-based strategy performs better in reducing overshoots when compared with static shedding. This is further supported by the deadline miss ratio measured: deadline miss events are abundant in static shedding while very few are observed in control-based shedding. Control-based shedding shows less undershoot than static shedding in both Fig 6a and Fig 6b.

6 Conclusions and Future Work

This paper presents the first attempt to use a proportional-integral controller to solve a practical problem - auto adaptation of stream qualities - in DSMSs. We argue that continuous queries in DSMSs are quality-critical in terms of timeliness, reliability, and precision. We propose a quality adaptation framework that emphasizes maintaining low levels of deadline misses. A common practice for DSMSs to overcome excessive incoming requests is load shedding, which is made difficult by the bursty data input and imprecise tuple processing cost estimation. We use a feedback control loop to dynamically adjust load under such uncertainties. Compared to previous work, the control-based approach leads to significantly better quality with less waste of resources.

We are currently in the process of implementing a quality controller in a real DSMS. This involves modeling and controlling more complex systems than the one described in this paper. Furthermore, we are investigating the possibilities of using control theory to solve other database problems such as data replication and dynamic resource allocation.

References

1. Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., Zdonik, S.: Monitoring Streams - A New Class of Data Management Applications. In: Procs. of the 28th VLDB Conf. (2002) 84–89
2. Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Motwani, R., Nishizawa, I., Srivastava, U., Thomas, D., Barma, R., Widom, J.: STREAM: The Stanford Stream Data Manager. IEEE Data Engineering Bulletin **26** (2003) 19–26
3. Chandrasekaran, S., Deshpande, A., Franklin, M., Hellerstein, J., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, M.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In: Proceedings of 1st CIDR Conference. (2003)
4. Olston, C., Jiang, J., Widom, J.: Adaptive Filters for Continuous Queries over Distributed Data Streams. In: Proceedings of ACM SIGMOD '03. (2003) 563–574
5. Tatbul, N., Çetintemel, U., Zdonik, S., Cherniack, M., Stonebraker, M.: Load Shedding in a Data Stream Manager. In: Proceedings of the 29th VLDB Conference. (2003) 309–320
6. Zhang, M., Madhyastha, T., Chan, N., Papadimitriou, S., Faloutsos, C.: Data Mining Meets Performance Evaluation: Fast Algorithms for Modeling Bursty Traffic. In: Proceedings of the 18th ICDE Conference. (2002) 507–516
7. Nahrstedt, K., Steinmetz, R.: Resource Management in Networked Multimedia Systems. IEEE Computer **28** (1995) 52–63
8. Arasu, A., Babcock, B., Babu, S., Datar, M., Rosenstein, J., Ito, K., Nishizawa, I., Widom, J.: Query Processing, Resource Management, and Approximation in a Data Stream Management System. In: Procs. of 1st CIDR Conf. (2003)
9. Babcock, B., Datar, M., Motwani, R.: Load Shedding for Aggregation Queries over Data Streams. In: Procs. of ICDE Conf. (2004)
10. Lu, C., Stankovic, J., Tao, G., Han, S.: Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms. Journal of Real-Time Systems **23** (2002) 85–126
11. Babcock, B., Babu, S., Datar, M., Motwani, R.: Chain: Operator Scheduling for Memory Minimization in Data Stream Systems . In: Proceedings of ACM SIGMOD '03. (2003) 253–264
12. Cheng, R., Kalashnikov, D., Prabhakar, S.: Evaluating Probabilistic Queries over Imprecise Data. In: Proceedings of ACM SIGMOD '03. (2003) 551–562
13. Franklin, G.F., Powell, J.D., Workman, M.L.: Digital Control of Dynamic Systems. Edison-Wesley, Massachusetts (1990)
14. Paraskevopoulos, P.N.: Modern Control Engineering. Marcel Dekker, New York (2002)
15. Abdelzaher, T., Sharma, V., Lu, C.: A Utilization Bound for Aperiodic Tasks and Priority Driven Scheduling. IEEE Trans. on Computers **53** (2004) 334–350