A Workload-Aware Adaptive Implementation of a Data Summarization Algorithm

David Chiu Qian Zhu Fatih Altiparmak Gagan Agrawal

Department of Computer Science and Engineering
The Ohio State University, Columbus OH 43210
{chiud, zhuq, altiparm, agrawal}@cse.ohio-state.edu

Abstract. We present a case study of versatile data stream summary construction on top of GATES (Grid-based AdapTive Execution on Streams), a middleware which enables grid-based processing of distributed data streams. An important characteristic of GATES is *self-adapting*, that is, it can enable an application to achieve its best accuracy while maintaining *real-time* constraints. The middleware's adaptive capabilities enable the development of workload-aware applications. This paper describes the implementation of an adaptive version of a multiple data stream summarization technique, namely PQ-Stream. The processing time and rate of compression achieved by this scheme is based directly on the number of bits utilized for quantization. We hypothesized that a self-regulated adjustment of the number of bits used for quantization could help manifest a robust data summary under stressful computing environments (e.g. massive number of streams or high data production rates). Our experimental evaluation shows that self-adaptation through a middleware is effective, and furthermore, depending on workload, a flexibly accurate synopsis can be constructed for summarizing multiple data streams.

1 Introduction

A growing number of applications across computer science and other science and engineering disciplines rely on, or can potentially benefit from, analysis and monitoring of data streams. Such applications include sensor data, call records, and weather nowcasting, to name a few. In the streaming model, data arrives continuously and must be processed in *real-time*, that is, the processing rate must match the arrival rate. Indeed, the essential challenge with data stream processing lies within its continuous nature: "Once an element from a data stream has been processed it is discarded or archived - it cannot be retrieved easily unless it is explicitly stored in memory, which typically is small relative to the size of the data streams" [2]. The portion of the streaming data that is maintained in memory, called a sliding window summary, is the data that have most recently arrived, and is generally the most relevant to the application.

Many streaming environments must also deal with the possibility of multiple data sources, as depicted in Figure 1. The massive amounts of incoming data imposes a heavy burden on the processing server, which may be further exacerbated if the data arrival rates are fast. To achieve some order of timeliness under such circumstances, faster processing and efficient storage management is imperative.

In our previous work, we developed a middleware, GATES (Grid-based AdapTive Execution on Streams), for enabling grid-based processing of distributed data

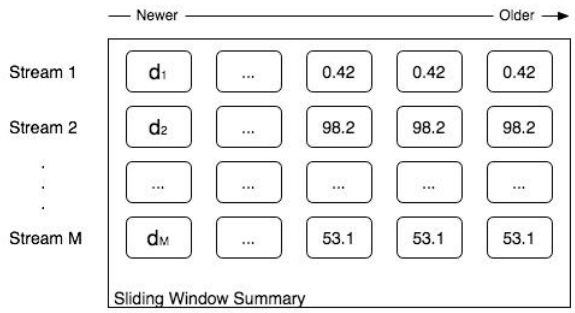

Fig. 1. A Multiple Data Stream System

streams [8, 7]. One important aspect of GATES is that it can enable an application to achieve the best accuracy, while maintaining *real-time* constraints. For this, the middleware allows the application developers to expose one or more *adaptation* parameters. An adaptation parameter is a tunable parameter whose value can be modified to increase the data processing rate, which would consequently reduce the accuracy of the processing in most cases. Examples of such adaptation parameters are: rate of sampling (e.g. what fraction of data items are actually processed), size of data summary structure, etc. The middleware automatically adjusts the values of these parameters to meet the real-time constraint on processing, through a *self-adaptation* algorithm, which is described and evaluated in full in [8, 7]. We have shown that, through using self-adaptation, our middleware can enable development of applications that are *resource-aware* and *workload-aware*.

This paper describes the design and implementation of an adaptive multi-stream summarization technique over GATES. We first select a proven multi-stream summarization method, PQ-Stream [5] (more on this later). Next, we investigate ways to expose an adjustment parameter within PQ-Stream's algorithm that will directly affect its summary accuracy and respective processing speed. Using our middleware as the grounding, and through an exploited form of PQ-Stream, we show that we can construct a fully adaptive and workload-aware multi-stream processor.

The rest of the paper is organized as follows. In Section 2, we summarize the PQ-Stream algorithm. In Section 3 we briefly describe the GATES middleware, and then highlight the main issues in our implementation. Our system is evaluated in Section 4. In Section 5, we discuss various related works, and conclude in Section 6.

2 PQ-Stream

PQ-Stream [5] is an efficient approach for sliding window summary construction over multiple data streams. This method extends predictive quantization (PQ) to multiple streams and proposes structures for real-time summarization and querying of a massive number of streams. We now briefly discuss the PQ-Stream technique, and then show why GATES serves as a natural foundation to support this process.

Prediction has been widely employed in applications such as multimedia and speech encoding. A prediction function f estimates the value of the sample $x[t]$, for dimension t, using previous samples, i.e.,

$$\tilde{x}[t] = f(x[t-1], x[t-2], \ldots)$$

The simplest type of prediction is performed by taking a linear combination of previous K elements, where K is the size of the prediction window,

$$\tilde{x}[t] = \sum_{k=1}^{K} p_k x[t-k]$$

is called *linear prediction*. Each p_k is known as the *prediction coefficient*, which is computed in a least squared errors manner so as to minimize the prediction error.

After making the prediction, the error of this prediction, $e[t]$, is calculated as

$$e[t] = x[t] - \tilde{x}[t].$$

This error, due to its narrow range, is an excellent candidate for a well-known lossy compression technique known as quantization.

Quantization can be defined as the process of reducing a large number of values by restricting them into a discrete set of intervals. These intervals are labeled by the average of all values that fall within. Thus, the actual values can be discarded, and only a mapping to their corresponding interval is stored. This stored mapping, along with these interval labels are usually much smaller than the original data summary. The rate of compression is based directly on the number of bits utilized for quantization. For instance, given a set of M values $D = \{d_1, d_2, \ldots d_M\}$, a b-bit quantizer allows us to compute 2^b intervals. Then each given data value, $d_k \in D$, is replaced with a b-bit mapping to the interval label that it most closely resembles. Thus, a larger value of b corresponds to a larger set of intervals, which translates to better data representation and poorer compression. In the extreme case where b is chosen to be $\lceil \log_2 M \rceil$, then the quantization will yield perfect data reconstruction. However, in most cases, since $2^{\lceil \log_2 M \rceil} \gg M$, it would make little sense to use this configuration as the "compression" will actually result in more storage utilization. In PQ-Stream, the set D refers to the set of incoming data elements from all M streams at a given time, as shown in Figure 1. We utilize Lloyd's method [10] for quantization, which is a classic heuristic to find the intervals that converge to a local minimum of errors.

3 GATES Middleware and Adaptive PQ-Stream

In this section, we initially describe the GATES middleware, and then our implementation of adaptive PQ-Stream.

3.1 Overview of the GATES Middleware

GATES (Grid-based AdapTive Execution on Streams) is a middleware that supports the flexible and adaptive analysis of distributed data streams. A key goal is to be able to allow the most accurate analysis while still being able to meet real-time constraints. For this purpose, GATES applies a *self-adaptation* algorithm. In summary, GATES has the following features:

– It is designed to use the existing grid standards and tools to the extent possible. Specifically, GATES is built on the Open Grid Services Architecture (OGSA) model and uses the initial version of Globus Toolkits (GT) 3.0's API functions. Therefore, all components of GATES, including applications, exist in the form of Grid services.

- It supports distributed processing of one or more data streams, by facilitating applications that comprise a set of *stages*. A stage in this context is a logical abstraction of a component in the grid execution sequence. For analyzing more than one data stream, at least two stages are required. Each stage accepts data from one or more input streams and outputs zero or more streams. The first stage is applied near sources of individual streams, and the second stage is used for computing the final results. Based upon the number and types of streams and the available resources, a GATES application developer might also specify more than two stages for facilitating intermediate results. GATES' APIs are designed to allow the specification of such stages.
- It flexibly achieves the best accuracy that is possible while maintaining the real-time constraint on the analysis. To do this, the system monitors: the producing rate at each data source, the available computing resources and local memory, and the available network bandwidth. Then it automatically adjusts the level of accuracy in the analysis by tuning the application's adaptive parameter within a certain range specified by the user.

Although the exact *self-adaptation* algorithm used in GATES falls beyond the scope of this paper, it has been evaluated extensively using a number of stream-based data mining applications, including counting samples and finding frequent itemsets in distributed data streams, using data stream processing for computational steering, and clustering evolving data streams. Results from the evaluation show that GATES is able to self-adapt effectively, and achieve the highest accuracy possible while maintaining the real-time processing constraint, regardless of the resource availability, network bandwidth, or processing power. Both the self-adaptation algorithm and its evaluation are described in full detail in [8, 7].

3.2 Adaptive PQ-Stream Implementation

The motivation behind implementing PQ-Stream on top of GATES is an intuitive one. Because GATES serves as a versatile middleware, it must support any variety of stream applications where timeliness is a must. Given heavy workload conditions (i.e. large number of streams and high data production rates), we must somehow adapt to this kind of situation by speeding up processing by expending some margin of accuracy on the observed data. In our implementation, this corresponds to using less bits for quantization. On the other hand, when workload is low, our system can afford to utilize more bits to obtain a more accurate sliding window summary. Thus, the number of bits is a natural performance parameter to be exposed for GATES manipulation. The advantage implementing an adaptive version of PQ-Stream is that it provides a way to construct a flexibly accurate sliding window summary which is workload sensitive.

The application's conceptual model is a simple one. The two-stage model consists of the *producer* and the *collector*. The *producer* allows any number of streams to generate data, which is sent to the *collector*, where PQ-Stream will quantize this data and store its synopsis as the sliding window summary. In terms of practice, the *producer* simulates the simultaneous existence of M data streams, and the *collector* acts as the central processing server. To simulate a streaming environment of M data streams, the *producer* first generates M data elements. This process then sleeps in proportion to its data production rate in order to simulate network latency, which can be calculated as follows: let K denote the size of data (in bits) that is to be

transferred, and let R bps (bits/sec) denote the actual physical network bandwidth. Then to simulate a synthetic bandwidth of C bps where $R \geq C$, we compute the delay as $latency = K(1/C - 1/R)$ sec. Notice that we subtract the difference of $1/R$ in order to compensate for the actual transfer time. It is easy to see that f our physical interconnect is fast (i.e. R is large), then this computation reduces to its familiar form of $latency = K/C$ sec.

In our system, a single streaming data element is 64 bits (akin to double-precision floating point numbers), and the physical bandwidth between two nodes in our cluster is 1 Gbps, then a $C = 50000$ bps simulated transfer rate corresponds to a *producer* delay of 1.27993 milliseconds. After the delay is observed, all M elements are sent to the *collector* to resemble M parallel data streams. We emphasize that the size of data is simply $K = 64$ bits, and not $K = 64 * M$ bits, in order to preserve the mimicry of sending M data elements simultaneously. After receiving all M data elements, the *collector* first deallocates the M oldest elements from the sliding window summary, then uses a b-bit quantization on the incoming data, where b is a suggested value obtained from GATES corresponding to the current work environment.

4 Performance Evaluation

This section presents an evaluation of our system. In the experimental setup, we use a dataset which consists of a varying number of synthetic data streams producing 4310 elements each. For each of our experiments, we used $M = 500$ and $M = 1000$ synthetic streams, and executed with producing rates of $50kbps$, $100kbps$, $200kbps$, and $400kbps$. Given that the maximum value of the PQ-Stream's adjustment parameter, b, is $\lfloor \log_2 M \rfloor$, we state that the range of b is $[0 \ldots 8]$ when $M = 500$ and $[0 \ldots 9]$ when $M = 1000$. We are particularly interested in showing the following:

1. The consistency of GATES in tuning the adjustment parameter.
2. The quickness of our system's adaptation to workload conditions.
3. The effects of adaptation on PQ-Stream's data synopsis construction.

For presenting these results, we define a *step* to refer to each time that a block of M data elements (from their respective data sources) arrive and are processed by the *collector*. We did not choose to depict our results compared against real time because we feel that the concept of a *step* corresponds better in this context. It allows us some insight into the system, since it is also at each *step* that our application asks GATES for the suggested parameter value. In our experimental setup, each stream produces exactly 4310 elements — this means the entire execution would contain a total of 4310 steps. Of course, in practical data stream applications, these numbers would be unbounded.

We also need to differentiate between "optimal setting" and "maximal setting" with respect to b. The maximal setting of b refers to the highest possible value that b can ever attain over the execution. We noted that the maximal setting is observed when $b = \lfloor \log_2 M \rfloor$, which can be achieved when the system is lightly loaded. The optimal setting of b is the highest possible value that b can attain under the given workload conditions to achieve timely processing. The misnomer of "optimality" is apparent when we claim, for instance, that $b = 1$ is optimal when the producing rate is $400kbps$.

4.1 Parameter Adjustment Consistency

These next experiments show that GATES persistently produces expected behavior. In Figure 2, we see that regardless of its initial value, b consistently converges to the optimal setting under the given workload. While these results were obtained from 500 streams each with producing rates of $50kbps$, we note that similar results were obtained on all other experimental modes (i.e. various producing rates and for $M = 1000$), not shown due to space restrictions. Although it is admittedly true that b does not necessarily converge to the maximal setting (that is, $b = \lfloor \log_2 M \rfloor$) in these followup experiments, it is however, expected due to various workloads.

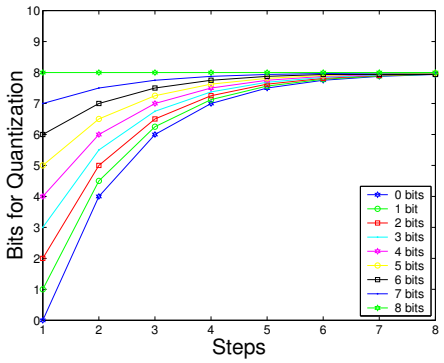

Fig. 2. Convergence of b given various initial values over $M = 500$ streams at 50kbps

4.2 Speedy Parameter Convergence

Referring back to Figure 2, we see that the convergence happens very quickly, taking only 8 steps (out of 4310).

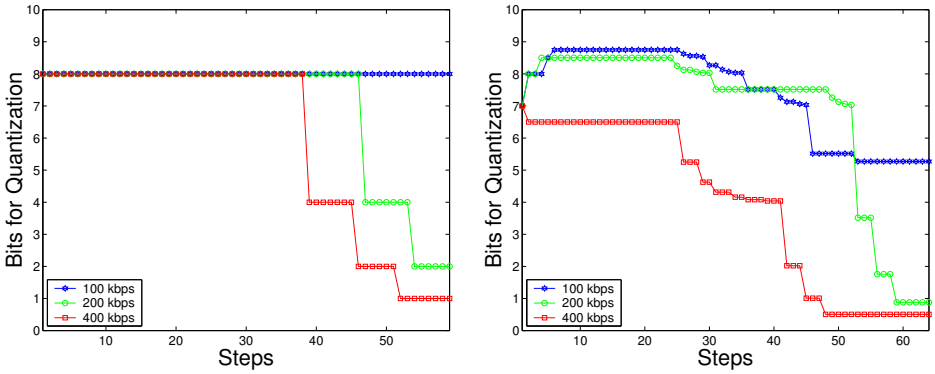

Fig. 3. Convergence of b over various production rates with (a) $M = 500$ streams and (b) $M = 1000$ streams

Figure 3(a) and 3(a) depicts further convergence speed results that were obtained after running experiments with producing rates of 100$kbps$, 200$kbps$, and 400$kbps$. Again, we see that b rapidly converges to its respective optimal setting under these various data production rates.

4.3 The Effects of Adaptation on PQ-Stream

In order to evaluate GATES' effect on PQ-Stream's synopsis construction accuracy, at each step t, we obtain a sum of squared errors (SSE) for all M streams. The SSE of a particular step t, denoted SSE_t, is computed as

$$SSE_t = \sum_{k=1}^{M} (d_{k,t} - \hat{d}_{k,t})^2$$

where $d_{k,t}$ denotes the actual data element from the kth stream which arrived at step t, and $\hat{d}_{k,t}$ denotes the quantized (reconstructed) counterpart of $d_{k,t}$. Taking the ratio SSE_t/M would give us the average error observed per stream.

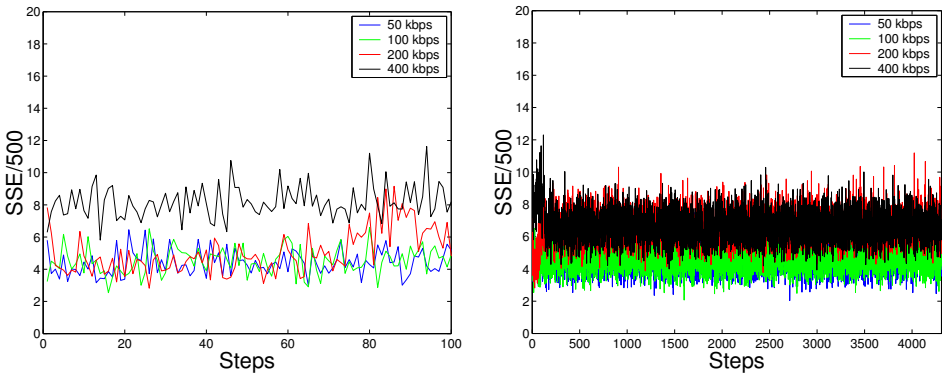

Fig. 4. SSE_t/M for $M = 500$ streams for (a) the first 100 steps and (b) the overall summary

First, we ran experiments for $M = 500$ streams. For ease of viewing the distinction, Figure 4(a) shows the resulting SSE_t/M of the first 100 steps. Figure 4(b) captures the ratio over the entire execution. Next, we ran the same experiments using $M = 1000$ streams. Because the amount of streams doubled, we would expect GATES to adapt correspondingly by reducing b in order to not only shorten processing time, but also compress the incoming data more vigorously. We see in Figures 5(a) and 5(b) that the average error observed per stream nearly doubled for all producing rates.

We mentioned earlier that using real-time in assessing these experiments would be unjustified. However, since we are claiming to construct a system that can meet real-time constraints, we show in Table 1 the execution times of the above simulations in their entirety. The results from this table show that increasing the amount of data streams under this system does not create a bottleneck the system. It is also clear from the average SSE observed per stream that the avoidance of system overloads

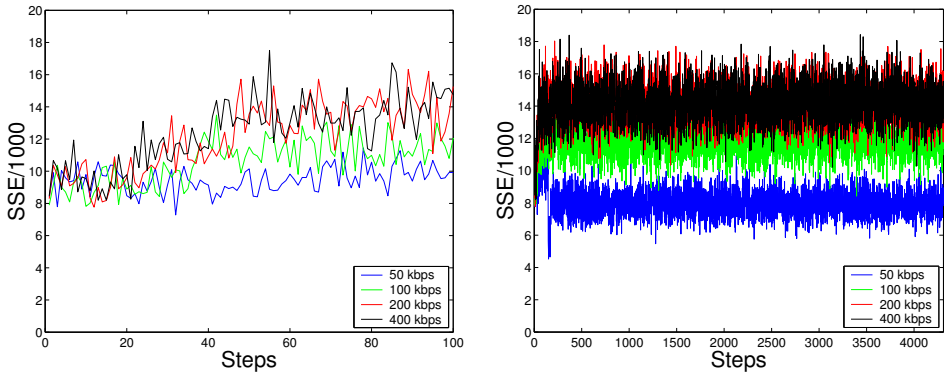

Fig. 5. SSE_t/M for $M = 1000$ streams for (a) the first 100 steps and (b) the overall summary

results in a less accurately represented data summary. This exemplifies the claim that our system is aware of the workload and can adapt itself to real-time restrictions. Of course, we accept this reification knowingly with a marginal loss of data accuracy.

Table 1. Execution times and average SSE of experimental trials

Streams	Producing Rate	Execution Time	Average SSE_t/M
500	50 kbps	175.483 sec	4.613
500	100 kbps	139.215 sec	4.616
500	200 kbps	106.525 sec	6.614
500	400 kbps	86.798 sec	6.681
1000	50 kbps	178.133 sec	7.998
1000	100 kbps	140.644 sec	11.826
1000	200 kbps	109.305 sec	14.169
1000	400 kbps	87.460 sec	14.173

5 Related Works

Our work reports the design and implementation of an adaptive multi-data stream processor. On one hand, we used our adaptive middleware, GATES, as a foundation to support the implementation. Application adaptation has been studied in many contexts, including as part of a grid middleware. Cheng *et al.* developed an adaptation framework [14]. Adve *et al.* [13] focused on language and compiler support for adapting applications to resource availability in a distributed environment. A number of projects also focused on operating systems, middleware, and networking support for adapting applications to meet Quality of Service (QoS) goals [15–18, 20]. SWiFT is a software feedback toolkit to support program adaptation [21]. However, it does not adapt the computation or support the notion of adaptation parameters. Conductor [19] is a middleware for deploying adaptation agents into a network. However, it does not adapt the computation and is not specifically designed for meeting

real-time constraints on processing. GATES is different in having runtime support for self-adaptation on computation to meet real-time constraint.

On the other hand, robust data stream systems have also received recent attention. Recall from before that there are two fundamental challenges of data streams: (1) real-time processing and (2) storage. The typical approach towards the first challenge is to alleviate CPU workload when possible, since an overloaded processor will not be able to keep up with the data arrival rate. Many *load shedding* techniques that drops portions of incoming data [9, 4, 11] have been proposed for such use. In addressing the second challenge, data reduction methods including sampling and sketching [3, 1, 6] can be used to deal with the limited storage issue. SPIRIT [12] is a system for efficiently monitoring multiple-data streams by reducing massive numbers of data streams into hidden variables which can capture trends and anomalies. The PQ-Stream approach [5] that we are exploiting is akin to SPIRIT in the way that it summarizes a multitude of data by maintaining some smaller variation in hopes that it can confidently represent the original data. These systems differ from load shedding schemes in the sense that they do not seek to drop portions of data from processing. Instead, they deal directly with manipulating the accuracy of all incoming data elements. It also should be noted that load shedding techniques can still be employed. Our system which consists of PQ-Stream over GATES is an integration of middleware adaptability and efficient multi-data stream management.

6 Conclusions

This paper described how we have used the GATES middleware to implement an adaptive version of PQ-Stream, which is a method for summary construction over multiple data streams. In implementing this algorithm using GATES, the number of bits for quantization is used as the adaptation parameter. GATES automatically adjusts the number of bits depending upon the workload, which includes the number of streams being processed and their producing rates. From our experimental evaluation, we saw that an adaptive PQ-Stream does indeed maintain a summary containing only marginal error. It is also clear from our experiments that these errors increase when the producing rate increases. This is due to the heavier workload observed on the server, and GATES' ability to supply some measure of adaptation under these stressful conditions. Given these results we claim that, by applying GATES adaptation to PQ-Stream, we can construct a workload-sensitive multi-stream summarization system.

References

1. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29, New York, NY, USA, 1996. ACM Press.
2. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *Proceedings of the 2002 ACM Symposium on Principles of Database Systems (PODS 2002) (Invited Paper)*. ACM Press, June 2002.
3. B. Babcock, M. Datar, and R. Motwani. Sampling from a Moving Window over Streaming Data. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*.

4. B. Babcock, M Datar, and R. Motwani. Load shedding for aggregation queries over data streams. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 350, Washington, DC, USA, 2004. IEEE Computer Society.
5. E. Tuncel, F. Altiparmak, and H. Ferhatosmanoglu. Incremental maintenance of online summaries over multiple streams. Technical report, The Ohio State University.
6. H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In *VLDB '98: Proceedings of the 24th International Conference on Very Large Data Bases*, pages 275–286, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
7. L. Chen and G. Agrawal. Supporting self-adaptation in streaming data mining applications. In *Proceedings of IEEE International Parallel & Distributed Processing Symposium(IPDPS)*, April 2006.
8. L. Chen, K. Reddy, and G. Agrawal. Gates: A grid-based middleware for distributed processing of data streams. In *Proceedings of IEEE Conference on High Performance Distributed Computing(HPDC)*, pages 192–201, June 2004.
9. N. Tatbul, U. Cetintemel, S. Zdonik, M. Chemiack, and M. Stonebraker. Load shedding in a data stream manager. In *Proceedings of VLDB '03*.
10. S. P. Lloyd. Least sSuares Quantization in PCM. In *IEEE Transactions on Information Theory*, pp 127-135, 1982.
11. Tu, Y., Liu, S., Prabhakar, S., and Yao, B. Load Shedding in Stream Databases: a Control-based Approach. In *Proceedings of the 32nd international Conference on Very Large Data Bases* Volume 32, pp 787-798, 2006.
12. S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming Pattern Discovery in Multiple Time-series. In *VLDB '05: Proceedings of the 31st international conference on Very Large Data Bases*, pages 697–708. VLDB Endowment, 2005.
13. Vikram Adve, Vinh Vi Lam, and Brian Ensink. Language and Compiler Support for Adaptive Distributed Applications. In *Proceedings of the SIGPLAN workshop on Optimization of Middleware (OM) and Distributed Systems*, June 2001.
14. Shang-Wen Cheng, David Garlan, Bradley Schmerl, Peter Steenkiste, and Ningning Hu. Software Architecture-based Adaptation for Grid Computing. In *Proceedings of IEEE Conference on High Performance Distributed Computing (HPDC)*. IEEE Computer Society Press, August 2002.
15. V. Bhargavan, K.-W. Lee, S. Lu, S. Ha, J. R. Li, and D. Dwyer. The TIMELY Adaptive Resource Management Architecture. *IEEE Personal Communications Magazine*, 5(4), August 1998.
16. Z. Jiang and L. Kleinrock. An Adaptive Pre-Fetching Scheme. *IEEE Journal of Selected Areas in Communication*, 1999.
17. B. Li and K. Nahrstedt. A Control Based Middleware Framework for Quality of Service Adaptations. *IEEE Journal of Selected Areas in Communication, Special Issue on Service Enabling Platforms*, 1999.
18. B. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. Walker. Agile Application-Aware Adaptation for Mobility. In *Proceedings of 16th ACM Symposium on Operating Systems Principles*, October 1997.
19. M. Yarvis, P. Reiher, and G. Popek. Conductor: A Framework for Distributed Adaptation. In *Proceedings of 7th Workshop on Hot Topics in Operating Systems*, 1999.
20. Ossama Othman and Douglas C. Schmidt. Issues in the Design of Adaptive Middleware Load-Balancing. In *Proceedings of the SIGPLAN Workshop on Optimization of Middleware and Distributed Systems*, pages 205–213. ACM Press, June 2001.
21. David C. Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole. A Feedback-driven Proportion Allocator for Real-Rate Scheduling. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*. ACM Press, December 1999.