Dynamic Stream Allocation with the Discrepancy between Data Access Time and CPU Usage Time

Sayaka Akioka
IT Research Organization
Waseda University
Tokyo, Japan
Email: akioka@aoni.waseda.jp

Yoichi Muraoka, and Hayato Yamana
Dept. of Computer Science and Engineering
Waseda University
Tokyo, Japan
Email: {muraoka, yamana}@waseda.jp

***Abstract*—Huge quantities of data arriving in chronological order are one of the most important information resources, and stream mining algorithms are developed especially for the analysis of the fast streams of data. A stream mining algorithm usually refers to the input data only once and never revisits them (read-once-write-once), while the conventional data intensive applications refer to the input data in a write-once-read-many manner. That is, once the stream mining falls behind, the process drops the input data until it catches up with the input data stream. Therefore, the fast execution of the stream mining leads the perfect analysis on all the input data, and it is very critical for the quality of the service. We propose a dynamic resource management for the stream mining in the distributed environment. The resource management utilizes the discrepancy between data access time and CPU usage time inside the stream mining, and speeds up the mining process. We implemented the methodology and proved successfully to process all the input data of such a fast data stream, whereas the serial execution drops more than 90% of the input data.**

***Keywords*-Load Balancing; Resource Management; Stream Mining.**

I. Introduction

A data stream, which is a sequence of data arriving in chronological order, is one of the significant information resources. The data streams include consumer generated media and mini blogs and many projects are trying to extract useful information through the analysis of the data streams. One of the technical obstacles for the scalable and real-time analysis of the data streams is their characteristic data access pattern. A stream mining algorithm is often utilized for the analysis of the data streams, and the stream mining algorithm refers to the input data only once. The input data themselves are never stored anywhere and never revisited. Additionally, the data stream rate is not consistent in the actual situation, and the computational cost of each stage is hard to clarify in advance. Therefore, this paper proposes a dynamic resource management focusing on the discrepancy between data access time and CPU usage time for the purpose of the practical speed-up of the stream mining algorithm in the highly distributed computational environment.

The rest of this paper is organized as follows. Section 2 introduces a stream mining algorithm, analyzes its model, and defines the problem to address in this paper. Section 3 describes the resource management methodology, and Section 4 discusses the effect of the resource manager through some validations in the actual cloud environment. Section 5 covers the related work, and Section 6 concludes this paper.

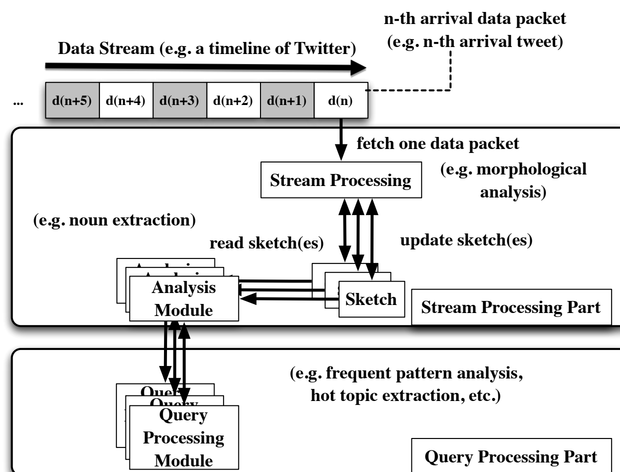

Fig. 1. A model of stream mining algorithms.

II. Problem Definition

A. A model of the stream mining algorithms

A stream mining algorithm is an algorithm specialized for a data stream analysis. Gaber et al. reviewed the theoretical foundations of the stream mining algorithms [1]. There are also many variations of the stream mining algorithms including clusterings [2]–[11], classifications [6], [12]–[16], frequency countings [17], [18], and time series analysis [19]–[24]. Regardless of the difference among the details of the stream mining algorithms, general stream mining algorithms share a fundamental structure and a data access pattern as shown in Figure 1, which Akioka et al. proposed [25].

A stream mining algorithm consists of two parts, which are the stream processing part and the query processing part. The major responsibility of the stream processing part is to process each data unit and extract the essence of the data for the further analysis by the query analysis part. The stream processing part needs to finish the processing of the current data unit before the next data unit arrives, otherwise, the next data unit will

Copyright (c) IARIA, 2012.

be lost as there is no storage for buffering the incoming data in a stream mining algorithm. On the other hand, the query processing part takes care of the further analysis such as a frequent pattern analysis and a hot topic extraction based on the intermediate data passed by the stream processing part, which a database system stocks usually. This clear difference between the responsibilities of these two parts indicates that only the stream processing part needs to run on a real-time basis. The successful processing of all the incoming data simply relies on the speed of the stream processing part.

The process flow for a single data unit in the stream processing part is as follows. First, the stream processing module picks the target data unit, and executes a quick analysis over the data unit, such as a morphological analysis and a word counting. Second, the stream processing module updates the data cached in one or more sketches with the latest results through the quick analysis. That is, the sketches keep the intermediate analysis, and the stream processing module updates the analysis incrementally as more data units are processed. Third and finally in the stream processing part, the analysis module reads the intermediate analysis from the sketches, and extracts the essence of the data, which is passed to the query processing part for the further analysis.

B. Motivation

There is a certain market requesting the complete analysis of one or more data streams on the fly. However, there is no solid solution for this problem targeting a large-scale, general stream mining algorithm yet. The model of a stream mining algorithm shown in Figure 1 indicates that the data access pattern of the stream mining algorithms are totally different from the data access pattern of so-called data intensive applications, which is intensively investigated in the high performance computing area. The data access pattern in the data intensive applications is write-once-read-many [26]. That is, the application refers to the necessary data many times during the computation; therefore, the key for the speed-up of the application is to place the necessary data close to the computational nodes for faster data access. On the other hand, in a stream mining algorithm, a process refers to its data unit only once, which is read-once-write-once style. Therefore, conventional techniques for the data intensive applications are not simply applicable for the speed-up or the complete analysis of the data stream.

III. METHODOLOGY

Following the discussion up to Section 2, there are two technical challenges to address. One of the challenges is the resource management for a scalable and complete analysis of the fast data stream. The other challenge is a generic framework for various stream mining algorithms. We propose a resource manager as a solution for these challenges, which enables execution of stream mining algorithms in the pipeline. The resource manager accepts data streams and scatters analysis tasks over the distributed computational environment such as the cloud and the grid. The resource manager also accommodates the number of the computational nodes to utilize dynamically according to the speed of the input data stream and the load of the analysis process. The rest of this section describes the details of the resource manager focusing on the stream processing part. As we already discussed in Section 2, only the stream processing part needs to run in a real-time manner.

A. Data Dependency and CPU occupancy

In a stream mining algorithm, one process of a data unit generates data dependency to the processes of the successive data units, and this is the major reason why the resource manager allocates the processes in a pipeline manner. As described in Section 2, the stream processing module updates the data in the sketches. That is, the sketches produce the data dependency across the processes. Figure 2 illustrates the data dependencies between two processes processing data units in line and the data dependency inside the process. More concretely, there are three data dependencies in Figure 2. One of the data dependencies is that the processing module in the preceding process should finish updating of the sketches before the processing module in the successive process starts reading the sketches (Dep.1 in Figure 2). Another data dependency is the processing module should finish updating the sketches before the analysis module in the same process starts reading the sketches (Dep.2 in Figure 2). The last data dependency is that the analysis module in the preceding process should finish reading the sketches before the processing module in the successive process starts updating the sketches (Dep.3 in Figure 2). These three dependencies are essential to keep the analysis results consistent and correct.

The data dependencies indicate that the access to the sketches reasonably divides the processing module into the two parts. That is, the stream processing module and the analysis module are reasonably managed independently, and these two modules in the same process are ideally allocated on the same CPU for the purpose to minimize the access cost to the sketches. Once we decide to allocate the processing module and the analysis module independently, the possible situation is that the processing module in one process occupies the CPU resource only waiting for the access to the sketch by the analysis module in the same process or the processing module in the successive process. This is the discrepancy between data access time and CPU usage time. The flexible resource management avoids this waste of the CPU resources. We revisit this problem and give the details in the next section.

B. Dynamic Resource Management

Following the discussion above, the resource manager allocates tasks taking care of both the three data dependencies and the discrepancy between data access time and CPU usage time. Here, the unit of the task is the former half or the latter half of the stream processing part, as shown in Figure 3.

The resource manager maintains three tables, which are the processing modules table, the analysis modules table, and the CPUs table, as shown in Figure 4 in order to manage the resources and tasks. Each entry of the processing modules

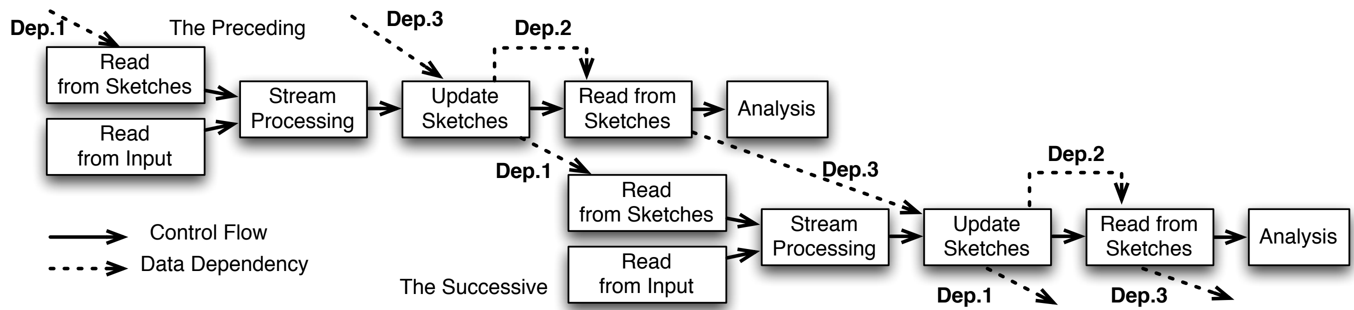

Fig. 2. The data dependencies between the two processes, which are processing two data units in line.

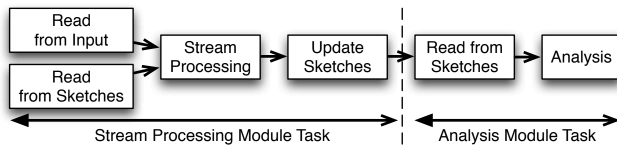

Fig. 3. The processing part consists of the two parts; the processing module and the analysis module. The resource manager recognizes each module as a task, and decides the resource allocation.

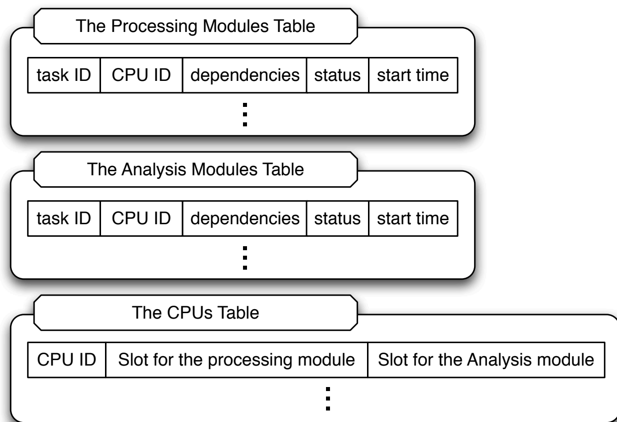

Fig. 4. The resource manager maintains the three tables; the processing modules table, the analysis modules table, and the CPUs table. These tables tell the resource manager the availability of the CPU resources, and the statuses of the ongoing tasks.

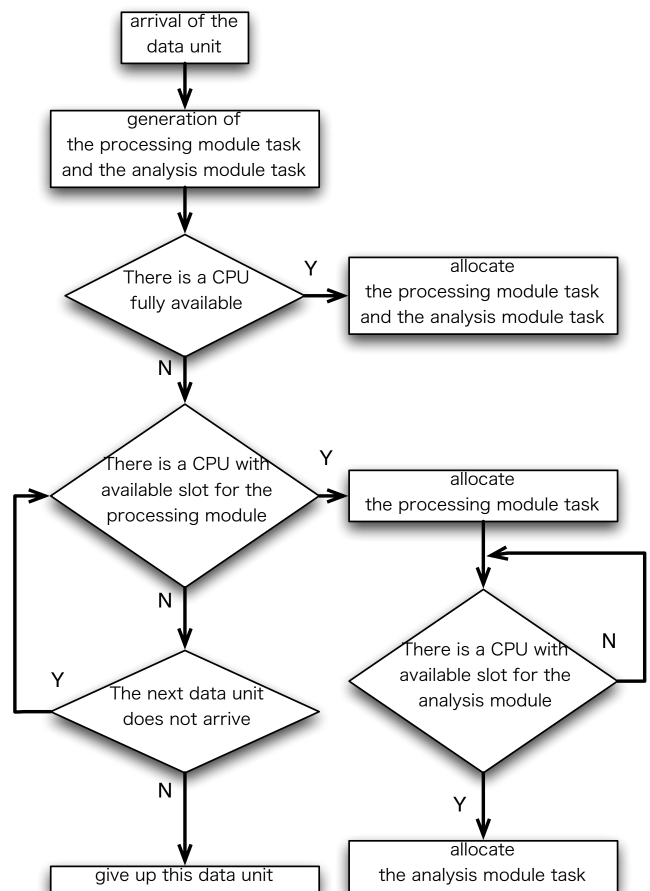

Fig. 5. The procedure of the resource manager.

table and the analysis modules table represents each task. The entry consists of the task id of this task, the CPU id where this task runs on, the task ids those this task has data dependency with, the start time of this task, and the current status of this task. Each field of the entry of the tables lets the resource manager know the data dependencies among the ongoing tasks, and whether the ongoing task occupies the CPU resource only waiting for the data access or not. The resource manager also maintains the CPUs table, and each entry of the table represents each CPU resource. Each entry consists of two fields, which are one field for the processing module task, and the other field for the analysis module task. The CPU table tells the resource manager the availability of CPU units.

Figure 5 illustrates the procedure of the resource manager. On the arrival of the data unit, the resource manager generates two tasks. One task is for the processing module, and the other task is for the analysis module. The resource manager generates a task id and an entry of the corresponding table in Figure 4. Once the processing module task and the analysis module task are ready, the resource manager decides the

TABLE I
THE SPECS OF THE COMPUTATIONAL NODES AND THE RESOURCE MANAGER.

Node Type	Num. of nodes	CPU+Memory
Node Type A	32 nodes	Xeon X5550+32GB
Node Type B	14 nodes	Xeon X5450+8GB
Node Type C	6 nodes	Xeon X5430+8GB
Resource Mngr.	1 node	Xeon E5520+24GB

resource allocation for these tasks. Here, there are three possible situations. First situation is that there is an available CPU whose slots are available both for the processing module and for the analysis module. In this case, the resource manager allocates the corresponding tasks onto this CPU, and returns to wait for the successive data units. Second situation is that there is an available CPU only the slot for the processing module is available. This time, the resource manager allocates only the processing module onto this available CPU, and waits until the slot for the analysis module on any CPU becomes available. As the processing module of this data unit never starts before the analysis module of the preceding data unit starts, the allocation of the analysis module is not urgent. Third and the worst situation is that there is no available slot for the analysis module. In this situation, the resource manager tries to find an available slot until the next data unit arrives, and finally gives up the corresponding data unit when no slot turns to be available.

IV. VALIDATION

We implemented the resource manager as a Java program and validated the effect of the resource manager in the actual computational cloud environment. We utilized 53 nodes of IBM Computing on Demand (CoD) as the cloud environment, and all the computational nodes are connected each other via Gigabit Ethernet. Table I summarizes the specs of the computational nodes of CoD. As shown on Table I, the computational environment was heterogeneous. However, this version of the resource manager does not consider the variations of the computational nodes, and the further functionality with the heterogeneous environment is reserved as future work.

The input data stream was generated inside of CoD under the assumption of a very fast data stream. More concretely, we took Twitter as a model of the input data stream. The data stream generator randomly chooses one post from the 1000 tweets pool, which is the actual sample from Twitter timeline. As the total number of the tweets per day is approximately 100 million, therefore, the tweet generator posts one tweet every millisecond in order to form a real Twitter like input data stream. We concluded this is the best way to simulate Twitter data stream, because collecting all the posts via Twitter API is practically almost impossible. Each computational node performs morphological analysis over the data unit in the processing module, and counts the appearances of the specific expressions such as URL or referred accounts in the analysis module. If either the processing module or the analysis module

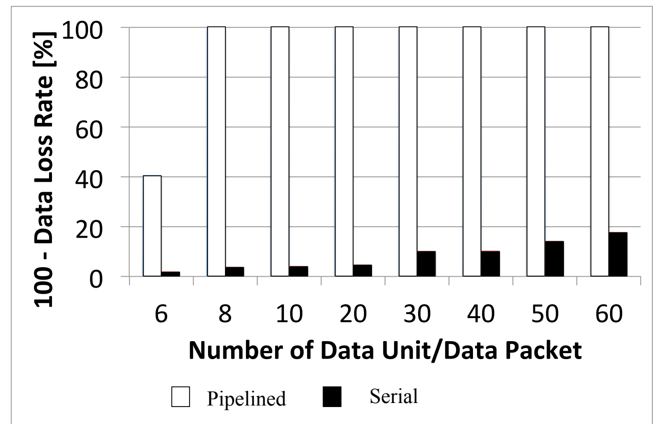

Fig. 6. The rates of the processed of the incoming data. The higher is better.

takes longer and results to break the dependencies discussed in Section 3, the corresponding data unit is given up as the analysis qualitywill not be ensured anymore. All the analysis modules are implemented with Java and Java RMI.

Figure 6 plots the rates of the processed of the incoming data units in comparison between the pipelined executions with the proposed resource manager and the serial executions. Here, let n_{total} be the total number of the incoming data units, and let $n_{processed}$ be the total number of the successfully processed data units among the incoming data units. Following these definitions, the rate of the processed of the incoming data units $R_{processed}$ is defined as follows.

$$R_{processed} = 100(1 - \frac{n_{processed}}{n_{total}})$$

In Figure 6, the white bars represent the rates of the processed of the incoming data units with the pipelined executions, and the black bars represent the rates of the processed of the incoming data units with the serial executions. The x-axis indicates the numbers of tweets packed into one data unit, and the y-axis indicates the rates of the processed of the incoming data units in percentile. The higher is better. Ideally, each tweet should be located to one computational node, however, this scenario was not effective even with all the available 52 nodes in the experimental environment this time; both the pipelined version and the serial version lost more than 80% of the incoming data. Therefore, we decided to group several tweets into a data unit, and let each computational node process each data unit. Figure 6 indicates that the pipelined version dropped about 60% of the incoming data with six tweets packed into one data unit. However, the pipelined version processed all the incoming data when one data unit contains eight or more tweets. The drop rate decreases if you pack more tweets into one data unit, because you have longer time slot for the larger data unit, and you have more relaxed deadline for the dependencies discussed in Section 3.

On the other hand, the serial execution always drops more than 80% of the incoming data regardless of the number of tweets contained into a data unit. For example, we observed the

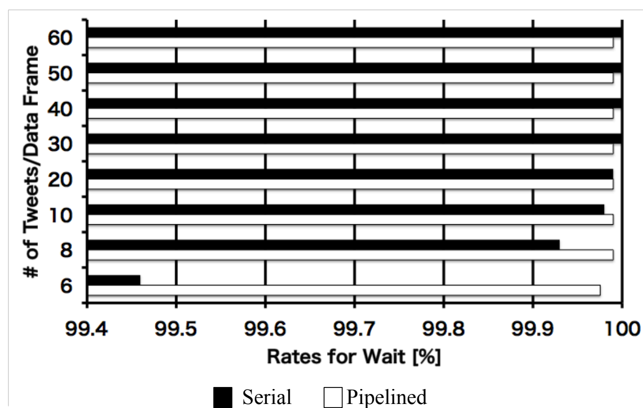

Fig. 7. The rates of waiting for the tasks for data dependencies.

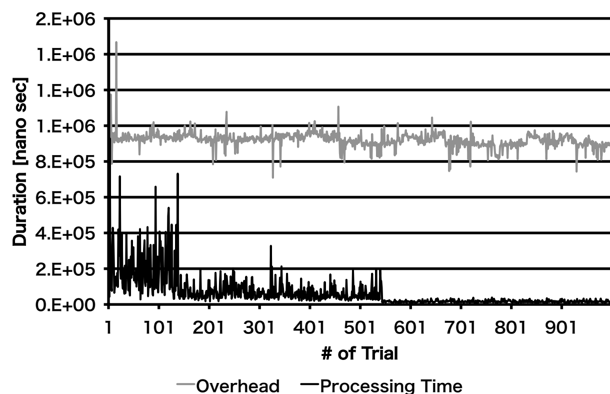

Fig. 8. Overheads of Java RMI calls and the actual time costs for the analysis processes on the incoming 1000 different tweets.

serial execution processed only less than 20% of the incoming data even with 60 tweets packed into one data unit. This situation is equivalent to the batch process of all the tweets every second, and the result indicates that the batch process invoked every second is not practical at all for the purpose of the realtime analysis of the actual fast data stream such as Twitter. The series of the experiments concludes that the pipelined execution is indespensable and effective for on-the-fly analysis of the realistic fast data streams.

Figure 7 plots the rates of the waiting in the pipelined execution with the resource manager. That is, the values in Figure 7 represent how many processes had to wait for another process to read from the sketch. The more idle tasks occupy more CPU resources as the rate for the wait increases. The x-axis represents the rates of the waiting and the y-axis represents the number of the tweets packed into one data unit. The black bars indicate the rates of the wait for the serial executions, and the white bars indicate the rates of the wait for the pipelined executions. Apparently, the pipelined tasks face the wait state more often than the serial tasks, which is natural because the pipelined version drops the incoming data less and tries to process all the incoming data. Figure 7, however, reveals that almost all the tasks had to wait for another task because of the data dependencies. We can assume that more tasks might be processed with less number of computational nodes once this situation is resolved and, therefore, we are planning to investigate this problem further in the future work.

We also recognize the experimental results shown here are possible to change according to the balance between the speed of the incoming data stream and the time cost for the analysis process on each tweet. Figure 8 shows the overheads of Java RMI calls and the actual times costs for the analysis processes on the incoming 1000 different tweets as a reference in order to give an overview of the balance of the time costs in the experiments in this section. In Figure 8, the gray line indicates the overheads of Java RMI calls, and the black line indicates the actual time costs for the analysis processes. The x-axis represents the number of the trials, and the y-axis represents the time costs in nano seconds. We also implemented a simulator to simulate the behavior of the resource manager in the distributed computational environment with the various settings of the speed of the incoming data stream, the time cost of the analysis process on each incoming data unit, and the overheads of the network communications and remote procedure calls. For the limitation of the space, we leave out the detailed results of the simulations and the further disucssion on the results here. We just add the simulation results showed quite similar tendency to the results already discussed in this section.

V. Related Work

The high performance computing area has been energetically developed serveral speed-up techniques targeting the applications with the huge input data [26]–[31]. Such an application is called the data intensive application. However, these techniques are not simply applicable for the stream mining because of the data access patterns. A data intensive application accesses data in a write-once-read-many manner, and, therefore, the speed-up techniques for a data intensive application is to mainly for the purpose of the speed-up of the access to the well utilized data. On the other hand, a stream mining usually refer to the input data only once and, therefore, the speed-up techniques for the data intensive applications do not work well.

There are several researches on the speed-ups of the stream mining algorithms through static resource management [32]–[34]. Those resource managers request a solid estimation for each stage of the computation, and also put an assumption that the input data stream rate is consistent. As the actual data stream is not consistent and the estimation of the computational cost in advance is practically impossible in the actual applications and, therefore, a dynamic resource management just as proposed in this paper needs to be developed.

VI. Conclusion

This paper proposed a dynamic resource management for the perfect analysis of the fast and realistic data streams

such as Twitter, focusing on the discrepancy between data access time and CPU usage time. The proposed methodology was implemented as the resource manager in the acutual cloud environment, and validated its efficiency. The results demonstrated that our approach is indispensable to process the fast data stream without a drop; otherwise, 80-90% of the incoming data will be lost.

ACKNOWLEDGMENTS

This paper is based in part on work supported by “Multimedia Web Analysis Framework towards Development of Social Analysis Software”, funded by Japanese Science and Technology Agency (JST).

REFERENCES

[1] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, “Mining data streams: a review,” *ACM SIGMOD Record*, vol. 34, no. 2, pp. 18 – 26, 2005.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, “A framework for clustering evolving data streams,” in *Proc. the 29th international conference on very large data bases (VLDB’03)*, 2003, pp. 81 – 92.

[3] ——, “A framework for projected clustering of high dimensional data streams,” in *Proc. the 30th international conference on very large data bases (VLDB’04)*, 2004, pp. 852 – 863.

[4] B. Babcock, M. Datar, R. Motwani, and L. O’Callaghan, “Maintaining variance and k-medians over data stream windows,” in *Proc. ACM SIGMOD/PODS 2003 Conference*, 2003, pp. 234 – 243.

[5] M. Charikar, L. O’Callaghan, and R. Panigrahy, “Better streaming algorithms for clustering problems,” in *Proc. the 35th annual ACM symposium on Theory of computing (STOC’03)*, 2003, pp. 30 – 39.

[6] P. Domingos and G. Hulten, “Mining high-speed data streams,” in *Proc. the sixth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD’00)*, 2000, pp. 71 – 80.

[7] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O’Callaghan, “Clustering data streams: Theory and practice,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 515 – 528, 2003.

[8] S. Guha, N. Mishra, R. Motwani, and L. O’Callaghan, “Clustering data streams,” in *Proc. the annual symposium on foundations of computer science*, 2000, pp. 359 – 366.

[9] G. Hulten, L. Spencer, and P. Domingos, “Mining time-changing data streams,” in *Proc. the seventh ACM SIGKDD international conference on Knowledge discovery and data mining (KDD’01)*, 2001, pp. 97 – 106.

[10] L. O’Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, “Streaming-data algorithms for high-quality clustering,” in *Proc. the 18th international conference on data engineering (ICDE2002)*, 2002, pp. 685–694.

[11] C. Ordonez, “Clustering binary data streams with k-means,” in *Proc. the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery (DMKD’03)*, 2003, pp. 12–19.

[12] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, “On demand classification of data streams,” in *Proc. the tenth ACM SIGKDD international conference on knowledge discovery and data mining (KDD’04)*, 2004, pp. 503 – 508.

[13] Q. Ding, Q. Ding, and W. Perrizo, “Decision tree classification of spatial data streams using peano count trees,” in *Proc. the 2002 ACM symposium on Applied computing (SAC’02)*, 2002, pp. 413–417.

[14] V. Ganti, J. Gehrke, and G. Ramakrishnan, “Mining data streams under block evolution,” *ACM SIGKDD Explorations Newsletter*, vol. 3, no. 2, pp. 1 – 10, 2002.

[15] S. Papadimitriou, A. Brockwell, and C. Faloutsos, “Adaptive, hands-off stream mining,” in *Proc. the 29th international conference on very large data bases (VLDB’03)*, 2003, pp. 560 – 571.

[16] H. Wang, W. Fan, P. S. Yu, and J. Han, “Mining concept-drifting data streams using ensemble classifiers,” in *Proc. the ninth ACM SIGKDD international conference on knowledge discovery and data mining (KDD’03)*, 2003, pp. 226 – 235.

[17] G. Cormode and S. Muthukrishnan, “What’s hot and what’s not: tracking most frequent items dynamically,” *ACM Transactions on Database Systems (TODS) - Special Issue: SIGMOD/PODS 2003*, vol. 30, no. 1, pp. 249 – 278, 2005.

[18] G. S. Manku and R. Motwani, “Approximate frequency counts over data streams,” in *Proc. the 28th international conference on very large data bases (VLDB’02)*, 2002, pp. 346 – 357.

[19] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang, “Multi-dimensional regression analysis of time-series data streams,” in *Proc. the 28th international conference on very large data bases (VLDB’02)*, 2002, pp. 323 – 334.

[20] V. Guralnik and J. Srivastava, “Event detection from time series data,” in *Proc. the fifth ACM SIGKDD international conference on knowledge discovery and data mining (KDD’99)*, 1999, pp. 33 – 42.

[21] J. Himberg, K. Korpiaho, H. Mannila, J. Tikanmaki, and H. Toivonen, “Time series segmentation for context recognition in mobile devices,” in *Proc. the 2001 IEEE International Conference on Data Mining (ICDM’01)*, 2001, pp. 203 – 210.

[22] P. Indyk, N. Koudas, and S. Muthukrishnan, “Identifying representative trends in massive time series data sets using sketches,” in *Proc. the 26th International Conference on Very Large Data Bases (VLDB’00)*, 2000, pp. 363 – 372.

[23] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, “A symbolic representation of time series, with implications for streaming algorithms,” in *Proc. the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery (DMKD’03)*, 2003, pp. 2 – 11.

[24] Y. Zhu and D. Shasha, “Statstream: statistical monitoring of thousands of data streams in real time,” in *Proc. the 28th international conference on very large data bases (VLDB’02)*, 2002, pp. 358 – 369.

[25] S. Akioka, Y. Muraoka, and H. Yamana, “Data access pattern analysis on streaming mining algorithms for cloud computation,” in *Proc. the 2010 International Conference on Parallel and Distributed Processing (PDPTA2011)*, 2011, pp. 36 – 42.

[26] I. Raicu, I. T. Foster, Y. Zhao, P. Little, C. M. Moretti, A. Chaudhary, and D. Thain, “The quest for scalable support of data-intensive workloads in distributed systems,” in *Proc. the 18th ACM international symposium on High performance distributed computing*, 2009, pp. 207 – 216.

[27] S. Doraimani and A. Iamnitchi, “File grouping for scientific data management: Lessons from experimenting with real traces,” in *Proc. the 17th International Symposium on High Performance Distributed Computing (HPDC’08)*, 2008, pp. 153 – 164.

[28] I. Raicu, Y. Zhao, I. T. Foster, and A. Szalay, “Accelerating large-scale data exploration through data diffusion,” in *Proc. the 2008 International Workshop on Data-aware distributed computing (DADC’08)*, 2008, pp. 9 – 18.

[29] S. Y. Ko, R. Morales, and I. Gupta, “New worker-centric scheduling strategies for data-intensive grid applications,” in *Proc. the 8th ACM/IFIP/USENIX international conference on Middleware (MIDDLEWARE2007)*, 2007, pp. 121 – 142.

[30] S. Venugopal and R. Buyya, “A set coverage-based mapping heuristic for scheduling distributed data-intensive applications on global grids,” in *Proc. the 7th IEEE/ACM International Conference on Grid Computing (GRID’06)*, 2006, pp. 238 – 245.

[31] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi, “Scheduling data-intensive workflows onto storage-constrained distributed resources,” in *Proc. the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid’07)*, 2007, pp. 14 – 17.

[32] B. Agarwalla, N. Ahmed, D. Hilley, and U. Ramach, “Streamline: a scheduling heuristic for streaming applications on the grid,” in *Proc. the 13th Annual Multimedia Computing and Networking Conference (MMCN’06)*, 2006, pp. 77 – 91.

[33] W. Zhang, J. Cao, Y. Zhong, L. Liu, and C. Wu, “An integrated resource management and scheduling system for grid data streaming applications,” in *Proc. the 2008 9th IEEE/ACM International Conference on Grid Computing (GRID’08)*, 2008, pp. 258 – 265.

[34] L. Chen and G. Agrawal, “A static resource allocation framework for grid-based streaming applications: Research articles,” *Journal Concurrency and Computation: Practice & Experience - Middleware for Grid Computing*, vol. 18, no. 6, pp. 653 – 666, 2006.