Real Time Business Performance Monitoring and Analysis Using Metric Network

Pu Huang, Hui Lei, Lipyeow Lim
IBM T. J. Watson Research Center
Yorktown Heights, NY, 10598

***Abstract*-Monitoring and analyzing business performance in a continuous manner nowadays is crucial for enterprises to achieve operational excellence, and to better align daily operations with long-term business strategies. To do so, performance measures need to be collected from daily operations and aggregated to construct higher-level Key Performance Indicators (KPIs) in nearly real time. We propose a system called metric network for enterprise-wide business performance monitoring and analysis. A metric network consists of metrics, metric repositories, aggregation agents, and knowledge agents. We describe in details the generic procedure patterns of these metric network entities and their communication pattern. Our loosely coupled design makes it easy to enhance features by adding more metrics and agents. The proposed approach is examined using real metrics on a fictitious scenario.**

I. INTRODUCTION

It is an often-quoted axiom of business that "you cannot manage what you cannot measure." Nowadays, enterprises have realized that monitoring and analyzing business performance in a continuous manner is crucial to achieve operational excellence, and to better align daily operations with long-term business strategies.

An enterprise executes various business processes in its every day operations. These processes often span several functional units; sometimes even extend to link with its partners' processes. They usually involve many different roles, assets, and resources; they may be supported by IT systems, or be executed in ad hoc manner manually. To monitor enterprise-wide business performance, one has to continuously collect metric data from these processes, aggregate the lower-level operational metrics to build higher-level Key Performance Indicators (KPIs).

In this paper, we present a system called metric network for enterprise-wide business performance monitoring and analysis. A metric network consists of metrics, metric repositories, aggregation agents, and knowledge agents. Metric repositories store metric values. These repositories are usually distributed, close to the business processes they are collecting data from. Aggregation agents automatically aggregate lower-level metrics to create higher-level KPIs in real time. This ensures that every day operational measures are reflected in the KPIs in a timely fashion, which is essential to make prompt executive decisions. Agents and metric repositories communicate through message passing, which makes the whole system loosely coupled, and ensures that it is easy to enhance features by adding more metrics and agents.

Metric measurements collected are not just for presentation. Our metric network also supports generic what-if analysis. In a what-if analysis, a user submits a hypothetical business scenario to a knowledge agent, which in turn responds with the estimated outcome of this scenario. What-if analysis is widely used in business to identify root causes, predict future performance, and evaluate strategy/operation changes. A key feature of our metric network is that it supports learning knowledge agents, which automatically build up learning models that are used to answer what-if queries.

Using a metric network, management at different level of an enterprise hierarchy can address their local concerns: they can use their own aggregation agents to build metrics to measure local performance; they can use their own knowledge agents to analyze business scenarios of their interest. They can do all these in a single metric network without interference with each other. However, since all share the same metric network, all this localized knowledge about how business is operated in a daily basis is integrated in the metric network through knowledge agents. This knowledge can be shared by the whole enterprise. Higher management can reuse the local metrics to monitor enterprise-wide performance, to do deeper what-if analysis by chaining up knowledge agents deployed at local level. In this sense, our metric network is an enterprise-wide knowledge integration tool.

II. RELATED WORK

Real time business process monitoring and data mining, and sensor network applications often generate a large amount of data continuously. These applications call for a new kind of data management system to persist, retrieve and query continuous data streams. Stream data management has drawn considerable attention in the past few years. Many issues that are unique to stream management have been addressed, including extending SQL to accommodate time-based and order-based operators, approximately answering queries, and accommodating long-run queries, etc. [2, 3, 9]. Many experimental systems have been built [1, 4, 5, 8, 7]. Although commercial systems are limited at the current stage, we expect many will be available in the near future as the technology and the market grow mature. Our metric network builds on top of stream management systems and relies on them to manage metric streams. A metric network is not an approach to manage stream metric data per se, but an approach to dynamically build meaningful relations between these streams.

One kind of emerging business applications that prompt stream management is real time business intelligence (BI). The promise of real time BI is to extract vital business information in nearly real time from streams of operational data generated by enterprises' daily operations. This requires real time stream management and real time (online) learning capabilities. Online learning algorithms are not necessarily different from offline algorithms. What matters is the

IEEE International Conference on e-Business Engineering (ICEBE'06)
0-7695-2645-4/06 $20.00 © 2006 IEEE

frequency an algorithm's internal learning model is updated. In the extreme case, it can be updated every time a new piece of training data is available. Doing so may not be necessary and may hurt performance. How frequent and under what conditions a learning model needs to be updated is application specific. Our metric network leaves this decision to the application designer. From a real time BI perspective, our metric network can be viewed as framework to build real time BI systems. Knowledge agents incorporate learning capabilities, and they interact with metric repositories (stream data managers) to enable real time BI.

,

III. METRIC AND METRIC REPOSITORY

We use the term "monitored object" to refer to any business entity/process whose performance is of the concern and thus is measured. A monitored object is usually measured by multiple metrics with each measuring a specific aspect of the object. Metrics directly measuring a monitored object are primitive metrics; metrics that are computed from lower-level metrics (primitive or derived) are derived metrics.

A metric network consists of metrics, metric repositories and agents. Each metric (primitive or derived) has a single repository, which is the place where all the historical metric values are stored. Each metric repository hosts a single metric. This one-to-one relation between metrics and metric repositories is adopted to make sure that the whole metric network design is conceptually simple. It is not an implementation requirement. For example, a traditional relational database can be used to host multiple metrics as long as it provides a metric repository interface for each metrics.

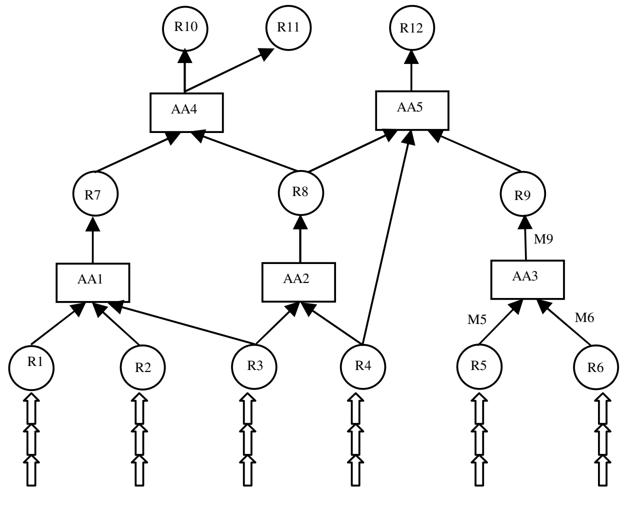

Figure 1: An example metric network.

There are two kinds of agents: aggregation agents combine lower-level metrics to create higher-level metrics; knowledge agents maintain the knowledge about the metrics for business analysis. Figure 1 shows an example metric network with 12 metric repositories (R1-R12) and 5 aggregation agents (AA1-AA5). Repositories R1-R6 store primitive metrics; they are generated by some external measure devices. Repositories R7-R12 store derived metrics; they are generated by aggregation agents. For example, metric M9 (corresponding to repository R9) is generated based on metrics M5 and M6 by agent AA3.

	Context Slot	Attribute
Metric Context Structure	Name	Daily Sales Revenue
	Slot 1	Store ID
	⋮	
	Slot N	Customer ID

Table 1: Contextual information of a metric.

			Instance 1	Instance 2	
Metric Data Stream Structure	Value				…
	Time				…
	Attributes	Store ID			…
		⋮			…
		Customer ID			…

Table 2: Structure of metric data instances

Any metric (primitive or derived) can be viewed as a stream of data generated by a measurement device: a primitive metric stream is generated by an external device; a derived metric stream is generated by an aggregation agent. A metric data stream consists of many metric instances; each instance represents the measurement result obtained from a single measurement activity. A metric repository is the place where the whole data stream is persisted.

Instance data need to be organized for easy access and query. Contextual information (meta-data) of a metric describes how its data instances are organized. A metric context consists of many slots; each slot contains an attribute of the metric. When designing a metric, one needs to decide what attributes need to be included, and put each attribute into a context slot. Any metric must have at least one attribute called Name; it represents the metric's name. Each metric should have a unique name (ID) within the whole metric network. Table 1 shows the contextual information of an example metric called "Daily Sales Revenue" with two attributes "Store ID" and "Customer ID". Metric contextual information is stored in a central meta-data store to facilitate system administration, see Section VI.

Each metric instance within a metric data stream contains three fields: Value, Time and Attributes. The Value field contains the measurement value of this instance; the Time field contains a timestamp of this instance, usually used to temporally correlate multiple metric instances; and the Attributes field contains the values of user-defined attributes in the context slots. These attributes are used to capture correlation information between metric instances. The values of all three fields (Value, Time and Attributes) in a metric instance will be assigned by an aggregation agent when generating this instance. (For primitive metric instances, those values are assigned by the external devices that generate them.) Table 2 shows the data stream structure of metric "Daily Sales Revenues" defined above.

In a metric network, metric repositories get metric instances generated by aggregation agents and store them; agents get metric instances from repositories to generate instances of high-level metrics (aggregation agents) or build metric relationship models (knowledge agents). All the repositories and agents follow a few well-defined procedures to communicate with each other.

Before entering into further discussion about these procedures, we first describe how entities (repositories and agents) in a metric network communicate with each other. If an agent wants to receive new instances from a metric repository, it needs to first register with this repository. Every time a new instance is received by the repository; it forwards the instance to all the agents in its registry list [1]. The same thing can be done for an aggregation agent, which may create new instances for multiple higher-level metrics; and each of these metric repositories needs to register with this aggregation agent. This communication pattern is called publication/subscription pattern. There are many pub/sub systems available, refer to Section VI for further discussion.

Here is the atomic procedure a metric repository executes after receiving a new metric instance.

Metric repository: store and forward an incoming new metric instance

1. *Store the received new instance into the repository.*
2. *Forward this new instance to all the agents in the registry.*

IV. AGGREGATION AGENT

Aggregation agents take metric instances from multiple lower-level metrics as input and generate one instance for each of its output metrics. They are long-living and autonomous entities. When there is no work to do, they go to sleep. Incoming messages wake them up to generate an output.

Two types of messages may wake up an aggregation agent: incoming new metric instances or time events. An incoming new metric instance may trigger the agent to generate an outgoing new instance. The agent has its own internal logic to decide whether a new instance will actually be generated or not. For example, agent AA3 in Figure 1 may create a new instance of metric M9 after receiving three instances of metric M5.

This implies that aggregation agents are long-living state machines. They are not simple stateless Web Services; they maintain their own internal states in responding to external events.

An aggregation agent may need to perform computation asynchronously. To see this, consider this example: a new instance of metric "Revenue" is generated every time a customer purchases a product. An agent takes metric "Revenue" as input and computes metric "Daily revenue" every day. To do so, the agent needs to set up a clock to wake up and do the computation.

Here is the generic procedure of aggregation agents.

Aggregation agent: check an incoming message and execute

1. *Check the incoming message to see whether it is a time event or a new metric instance. If it is a time event, go to 3; otherwise continue.*
2. *Check the incoming new metric instance to see whether new outgoing metric instances should be created. If yes, continue; otherwise, do some book keeping (update internal states) and go to step 6.*
3. *Get a list of output metrics.*
4. *Create a new instance for each of these metrics.*
5. *Forward each new instance to its corresponding repository.*
6. *Check whether there are more incoming messages, if yes, go to 1; otherwise go to sleep.*

An aggregation agent can apply the above procedure to create multiple temporally correlated output metrics. It can do so by simply marking the Time fields of these instances with the same timestamp.

By assign the Attributes fields of output metric instances with proper values, an aggregation agent can also correlate them according to an arbitrary user-defined logic. Further more, if the values in the Attributes fields of the output metric instances are assigned based on those of the incoming instances, an agent can also correlate the output metrics with input metrics.

[1] This can be implemented by sending a notification message instead of the actual new instance to the subscribers, and then the subscribers decide whether to retrieve the new instance at their own discretion. For conceptual simplicity, we describe the procedures as if all new instances are always forwarded. It should be clear that actually implementation may vary.

Most pub/sub systems guarantee that incoming new metric instances will be received as the same order as they were sent out. If this is not the case, the agent may need to cache incoming new instances and handle them at an appropriate time later. This can be done in step 2 where the agent executes the book keeping function.

V. KNOWLEDGE AGENT

Knowledge agents answer what-if queries. A what-if analysis is an interaction between a user and a knowledge agent: The user assigns hypothetical values to a group metrics, and feeds these values into a knowledge agent, which consequently returns its best estimate about the values of another group of metrics of interest. Since the output metric values from knowledge agents are estimates, sometimes information about how accurate these values are also attached, which is usually represented by probabilities. In general, a what-if analysis can be presented by the following process.

What-If analysis

1. *The user selects a group of metrics* $\{M_i\}$.
2. *For each metric* M_i*, the user creates multiple hypothetical instances* $\{v_{ij}\}$ *(assign values to* Value, Time *and* Attributes *fields), where* v_{ij} *is the j'th instance of metric* M_i.
3. *Feed all the instances* $\{\{v_{11}, v_{12}, ...\}, \{v_{21}, v_{22}, ...\}, ...\{v_{i1}, v_{i2} ...\}, ...\}$ *into a knowledge agent.*
4. *The knowledge agent produces a group of instance values* $\{w_{k1}, w_{k2}, ...\}$ *for each metric* Y_k *in a output metric set* $\{Y_k\}$*. It may also create a data set* $\{e_{k1}, e_{k2}, ...\}$ *that indicates how accurate the output metric values* $\{w_{k1}, w_{k2}, ...\}$ *are for each* Y_k.

A knowledge agent can be viewed as function $f(\cdot)$ mapping $V = \{\{v_{11}, v_{12}, ...\}, \{v_{21}, v_{22}, ...\}, ...\{v_{i1}, v_{i2} ...\}, ...\}$ to $W = \{w_{k1}, w_{k2}, ...\}$ and $E = \{e_{k1}, e_{k2}, ...\}$, i.e.,

$$W, E = f(V) \ .$$

This is synchronized mapping, meaning that given the inputs, the knowledge agent generates the outputs immediately. There is no time delay except the computation time.

There are two different approaches to implement a knowledge agent. One is to hard code function $f(\cdot)$ into the agent. In this approach, the logic of function $f(\cdot)$ needs to be known in prior; and once coded, this logic stays fixed. This approach is suitable when the relationship between the input metric sets $\{M_i\}$ and the output metric set $\{Y_k\}$ is known in prior and does not change frequently.

Another approach is to equip the agent with learning capabilities such that it can discover the function $f(\cdot)$ autonomously. The advantage of this approach is clear: 1) sometimes the mapping function $f(\cdot)$ between input and output metrics is unknown; and 2) it will allow what-if queries always being answered based on the most up-to-date knowledge.

To do so, a knowledge agent maintains a learning module. This learning module has a very similar structure as an aggregation agent: they both takes a group of metric as input, they both register to metric repositories to receive new metric instances of their interest, and they both are able to receive time events. The only difference is that instead of generate new metric instances as the aggregation agent does, the learning module within a knowledge agent always updates its internal model $\tilde{f}(\cdot)$, which is the current approximation of the unknown functional mapping $f(\cdot)$, based on its own updating logic.

Just like an aggregation agent, the learning module of a knowledge agent can get training data from metric repositories either passively or actively. In a passive pattern, the new metric instances are forwarded to the module once they are created. In an active pattern, the knowledge agent retrieves metric instances as it needs. When to retrieve data is determined by its own internal logic.

A knowledge agent is also a long-living autonomous entity, it continuously and incrementally update $\tilde{f}(\cdot)$ as more training data become available. It has three basic methods to control the updating frequency: 1) it can wake up to learn periodically under the control of a time clock, 2) it can update $\tilde{f}(\cdot)$ every time a new instance of an input metric is received, or 3) or it can pre-specify a rule and wake up every time this rule is triggered. The last method provides a great deal of flexibilities. For example, a knowledge agent can specify that it will wake up every time when receiving a metric instance with Value= 123. Just like aggregation agent, learning knowledge agents are long-living state machines. They maintain their own logic and states for learning.

When receiving a what-if query, the currently available approximation function $\tilde{f}(\cdot)$ is applied on the incoming query to generate an answer to it.

$$W, E = \tilde{f}(V)$$

This process of applying the learned function is executed a separate thread within a knowledge agent to ensure that two activities: learning the model, and applying the model, can run in parallel.

To summarize, a knowledge agent receives three types of messages: what-if queries, time events, and new metric instances. Here is the generic procedure for knowledge agents.

```
Knowledge agent: learning and applying the model

1. Is incoming message a what-if query? If yes,
   continue; otherwise, got to 3.
2. Start a new thread,
   a. Take the current learning model
      $\tilde{f}(\cdot)$,
   b. Output $W, E = \tilde{f}(V)$
   c. Exit this thread and go to step 7.
3. Check the incoming message to see whether it
   is a time event or a new metric instance. If it is
   a time event, go to 5 (start learning); otherwise
   continue.
4. Check the incoming new metric instance to see
   whether it is time to generate a new learning
   model. If yes, continue; otherwise, do some
   book keeping ((update internal states) and go
   to step 7.
5. Get all the training data required for learning.
6. Update learning model $\tilde{f}(\cdot)$.
7. Check whether there are more incoming
   messages, if yes, go to 1; otherwise go to sleep.
```

VI. Discussion

To facilitate administration and management, we require all entities in a metric network publish their meta-data in a single meta-data store.

1. The contextual meta-data of all the metrics should be published.

2. All metric repositories publish what metric it stores and data query interface (used to retrieve historical metric instances).

3. All aggregation agents publish what their input and output metrics are.

4. All knowledge agents publish the format of the what-if they answer, the format of the training data they need, and training data sources.

To add a new entity to a metric network, one must first publish all the meta-data of this entity into the meta-data store. A meta-data store can be implemented by different technologies, including databases, XML files, or even data files with proprietary formats. Many database products provide enhancement specifically for meta-data management. For example, IBM DB2 Cube Views provides an API to manage and access metadata stored in DB2.

Similarly, many different technologies can be used to implement metric repositories. Since metric repositories need to store all the history metric data, database technology is a preferred choice. As mentioned before, many metric repositories may physically locate on a single database, as long as the metric repository interface is provided.

Long-living aggregation agents are usually implemented as background processes or services that perform a user-specified aggregation function on the input metric instances. The complexity of the aggregation agent greatly depends on the complexity of the user-specified aggregation function, the number of input and output metrics and their synchronicity characteristics. Various optimization strategies can be used in the implementation of an aggregation agent. Existing techniques from query processing and optimization in data streams and continuous queries readily apply. A detailed exposition is beyond the scope of this paper and the reader should refer to [11] for an overview. The choice of what processing and optimization techniques to use also depends on how the real-time requirement is specified. In most enterprises, refresh rates in the order of seconds can be considered real-time as compared to daily or weekly extract-transform-load (ETL) data warehousing processes. In the case of mission critical KPIs where millisecond refresh rates are required, then processing techniques from data stream systems such as Gigascope [10] can be adopted in the implementation of the Metric Network system.

Knowledge agents are usually implemented as services, since many users may access the same agent to analyze different scenarios at the same time. The learning module within a learning knowledge agent can be implemented as background processes or services, just like an aggregation agent. Another implementation decision is the frequency of model update. Incremental update algorithms exists for many types of models (e.g. regression, neural networks, statistical models) and those can be used if the model is expected to change frequently. In most cases, the model is expected to change much less frequently than the data in the metric instances; hence periodic training to update the model is probably sufficient to meet business requirements and is probably more efficient.

Many pub/sub middleware products are available to support communication between metric network entities; for example, BizTalk, Websphere MQ, JMS, etc.

Temporally correlated metrics are usually generated by a single aggregation agent so that a single clock is used to mark the Time fields of these metrics. If one needs to temporally correlate multiple metrics that are generated by different aggregation agents, one has to synchronize the clocks of these agents, the standard approach is to utilize Network Time Protocol [6]. If only the relative order of these metric instances, not the absolute time, matters, assigning each instance a consecutively increasing order number, as an attribute, can also solve the problem.

VII. A scenario analysis

In this section, we use an example scenario to illustrate how to construct a metric network and use it for serviceability prediction. Metrics in this scenario are part of the real metrics a Fortune 500 company measures to monitor its inventory and sales. In this fictitious scenario, this company wants to evaluate the risk and benefit of expanding its Web sales practice. To do so, 5 derived metrics M5-M9 are defined. The meaning of these metrics is self-explanatory by their names (see Table 3). These metrics are built on top of 4 primary metrics M1-M4. Each row in the table represents a metric; each column is an attribute. We mark in the table by "x" to indicate which metric has which attributes defined.

Every time a customer order arrives, a new instance of metric "Sales revenue" will be generated. The Value field of this new instance carries the sales revenue of this order. This metric has four attributes "Fulfillment region" , "Order ID", "Customer sector" and "Sales channel", which carries information about where this order will be fulfilled, a unique order ID, which customer sector this order is from, and which sales channel it is from (Web is a sales channel).

From metric "Sales revenue", we can easily aggregate over appropriate attributes to get metrics "Growth of sales of Web business", "Buy frequency" and "Web revenue". To get metric "Sales ratio of priority/growing products", we need to know what products each customer order contains and what products are labeled as priority/growing products. We assume this information is available. Based on it, we can calculate this metric.

		Fulfillment region (A1)	Order ID (A2)	Product ID (A3)	Customer sector (A4)	Sales channel (A5)
M1	Sales revenue	X	X		X	X
M2	Inventory	X		X		
M3	Stock-out		X	X		
M4	On-time ship performance	X	X		X	
M5	Sales ratio of priority/growing products					X
M6	Growth of sales of Web business					X
M7	Buying frequency				X	X
M8	Web revenue				X	X
M9	Customer serviceability	X			X	X

Table 3: A scenario with user-defined metrics and attributes.

A single customer order may contain multiple products. If one product runs out of inventory, a new instance of metric "Stock out" is generated. Once all the products an order contains are shipped, a new instance of metric "one-time ship performance" is generated. To compute derived metric "Customer serviceability", we take metrics "Sales revenue" and "One-time ship performance" as the input; correlate them on attribute "Order ID"; and then aggregate on "Order ID" to generate a new instance of metric "Customer serviceability"

With this metric network deployed, the company can monitor its performance in nearly real time. Now suppose the management wants to do a what-if analysis on metric "Customer serviceability", specifically, they would like to know what this metric looks like in the next month as the spring sales season is closing. They have developed a good learning algorithm that can predict "Customer serviceability" accurately. They have deployed this algorithm as a knowledge agent and use the historical data of metrics "Sales revenue" and "Inventory" to train it continuously. Now the matter of finding out what the "Customer serviceability" value is for the next month is as simple as providing a set of hypothetical values of "Sales revenue" and "Inventory", which can be either generated randomly by a simulator or inputted by the user, and then feed them into the knowledge agent to obtain the predicted performance value.

VIII. CONCLUSION

Business performance monitoring is crucial to all enterprises. The traditional ETL data warehousing approach is limited in its ability to provide KPIs in a near real time fashion. In this paper, we propose a Metric Network System to manage KPIs and compute them in near real time. Our system consists of loosely coupled metrics, metrics repositories, aggregation agents, and knowledge agents that are easily extensible and can be implemented using any commercially available Service-Oriented Architecture (SOA) stack.

REFERENCES

[1] A. Arasu, S. Babu, and J. Widom. "The CQL ContinuousQuery Language: Semantic Foundations and Query Execution". Technical Report 2003-67, Stanford University, 2003.

[2] P. Bonnet, J. Gehrke, P. Seshadri. "Towards Sensor Database Systems". In Proc. 2nd Int. Conf. on Mobile Data Management. pp3-14. pages 3-14. 2001.

[3] D. Carney, U. Cetinternel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. Zdonik. "Monitoring streams|, A New Class of Data Management Applications". In Proc. 28th Int. Conf. on Very Large Data Bases, pp. 215-226. 2002.

[4] S. Chandrasekaran and M. J. Franklin. "PSoup: a system for streaming queries over streaming data". *In* VLDB Journal, August 2003.

[5] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, M. Shah. "TelegraphCQ: Continuous Data flow Processing for an Uncertain World". In Proc. 1st Biennial Conf. on Innovative Data Syst. Res, pp. 269-280. 2003.

[6] D. L. Mills. “Network Time Protocol (Version 3) Specification, Implementation and Analysis”. Rfc-1305, 1992.

[7] A. Lerner, D. Shasha. “AQuery: Query Language for Ordered Data, Optimization Techniques, and Experiments”.Technical Report 2003-836, New York University, March 2003.

[8] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, R. Varma. “Query Processing, Approximation, and Resource Management in a Data Stream Management System”. In Proc. 1st Biennial Conf. on Innovative Data Syst. Res, pp. 245-256. 2003.

[9] Y. Zhu and D. Shasha, "Statstream: Statistical monitoring of thousands of data streams in real time". In Proceedings of the 28th ACM VLDB International Conf. on Vary Large Data Bases, pp. 358-369, 2002.

[10] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. In Proc. ACM SIGMOD, pp. 262, 2002.

[11] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In Proc. of PODS 2002, June 2002