April 24, 2007

RT0728
Engineering Technology 8 pages

Research Report

A Hybrid Event-Processing Architecture based on the Model-driven Approach for High Performance Monitoring

Yosuke Ozawa, Teruo Koyanagi, Mari Abe, Liangzhao Zeng

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

A Hybrid Event-Processing Architecture based on the Model-driven Approach for High Performance Monitoring

Yohsuke Ozawa, Teruo Koyanagi, Mari Abe, Liangzhao Zeng

IBM Tokyo Research Laboratory,
Yamato-shi, Kanagawa-ken, Japan
E-mail :{ ozawaysk, teruok, maria}@jp.ibm.com
IBM T.J. Watson Research Center,
Yorktown Heights, New York, United States
E-mail:lzeng@us.ibm.com

Abstract

Business Activity Monitoring (BAM) is a concept for computer systems which support monitoring situations and the performance of business processes as they execute. The monitoring system needs to capture and to process numerous events while correlating them with other events stored in a database. If the database accesses become very frequent, read-only and write-only accesses can achieve high throughputs by applying well known solutions; a result cache and a batch update. However, the event processing of the monitoring includes both simultaneously. It is the case that is hard for the existing solutions to achieve high throughputs.

Here, we introduce a new architecture, HM-Hybrid, to process events for monitoring efficiently. It is a hybrid of a result cache and a batch update. And our model-driven approach makes it possible for them to work together in situations that are normally regarded as being unsuited for them. It processed 1207 events per second, about 43000 queries per second. It is 6 times better performance over a naive caching architecture in our experiments. The architecture seems useful for monitoring applications where there are many update and select queries.

1. Introduction

In modern management, it is increasingly necessary for managers to respond quickly to its dynamic and fast-changing business circumstances. Gartner [1] defines "Business Activity Monitoring" (BAM) as:

> *BAM is how we can provide real-time access to critical business performance indicators to improve the speed and effectiveness of business operations. Unlike traditional real-time monitoring, BAM draws its information from multiple application systems and other internal and external (inter-enterprise) sources, enabling a broader and richer view of business activities. As such, BAM will be a natural extension of the investments that enterprises are making in application integration. It calculates Key Performance Indicators (KPI) by capturing events which is generated along with execution of business operations, and evaluates whether the KPIs meet the user defined business goals or not.*

As described above, BAM is based on the integrated applications, because measuring the performance of today's enterprise is not possible without considering it as a cooperation of many of its sub-businesses. Service Oriented Architecture is a key to integrate the enterprise applications [2]. The enterprise applications are published as Web Services, and the business processes described by BPEL [3] integrates them. The uniformed interface conformance of Web Services enables composition and execution of business processes and such a unified system environment allows developing a monitoring system. For the concept of BAM, as many as possible related systems should be connected with the process engine. And richer information about broader business activities is calculated from numerous events generated by the connected systems.

However, we face performance problems in processing huge numbers of events. In order to understand business situations, the monitor correlates each event with other events, following rules described in an observation model or other rule descriptions [4, 5]. For persistence, the temporary states of calculations have to be saved in a database. This means at least one select query for the correlation and one update query of a calculated KPI are required for monitoring. Huge number of read and write accesses to the database will occur simultaneously when thousands of events are processed. For our clients, such as manufacturers, stockbrokers, and security companies, these systems generate thousands of events each second.

The monitor must have enough performance to process such large number of events.

Such requirements are hard to satisfy with existing architectures. However, if the database accesses become very frequent, read-only and write-only accesses to the database can achieve high throughputs by applying well known solutions. Read-only accesses can be accelerated with simple cache solutions. The results of select queries can be cached in the application server in a result cache, and the cached data need never expired, because no modification will be attempted to the original data. Write -only accesses can be accelerated with batch solutions. Batch update collects the write requests over an interval, and then executes them asynchronously as a large, fast job. This is known to be extremely efficient for write accesses using database management systems.

However such solutions fail when read and write accesses to a database are mixed. There are no efficient general solutions for the mixed read-write cases. In the mixed case, batch update cannot be used because of its asynchronous characteristics. The monitor always needs the latest data to process the next event. The event must be correlated with the very recent events because of the real-time requirements for monitoring.

Also the result cache is inefficient. When write accesses occurred frequently, the result cache cannot be simply updated like write-through cache, because the result cache stores search result of SQL select statements. Therefore, the cached results may be invalidated at very short intervals and the cache hit ratio will be low.

Rather than separate solutions, our idea is to combine them. Batch update can defer updates, as long as fresh results can be retrieved from the result cache. This is possible by maintaining result sets in the cache with reflecting the update queries without accessing the base tables. This requires application information about any constraints on the data access patterns. The model-driven approach we used can provide such information.

Here we introduce HM-Hybrid, a new architecture to process events for monitoring systems. It is a hybrid of a kind of result cache with batch updates to a database. It automatically and efficiently maintains a result cache and this result cache allows batch update by responding to the monitor's queries as a substitute for the unsynchronized state of database.

In the second section, the position of our work is described with related works. In the third section, we describe the concept and architecture of HM-Hybrid including the model-driven approach. The fourth section describes our evaluation of the architecture. In our experiment, HM-Hybrid realizes six times higher performance over a naive cache architecture. In the last section, we conclude that our architecture is quite effective to process high volume event streams, such as the monitoring components in BPM services.

2. Related works

Recently, several monitoring systems have been developed. Baresi et al. [6] illustrates a monitoring solution for composed services described as BPEL processes. They present two implementations. One is a dynamic link approach based on an object-oriented language, and the other is an interpreted approach based on an assertion rule language. Mahbub et al. [7] also shows an implementation of a monitoring system on an execution engine for BPEL. It captures the events generated by the engine, and detects whether or not the requirements described in a rule language are satisfied in running business processes. Zeng et al. [4] introduced a compilation approach to the model-driven monitoring system. It showed that an observation model could be transformed into a simple executable model based on SQL.

These systems save monitored states in databases. As mentioned above, batch update is extremely efficient to process high volumes of events. However, the latency of the batch update constitutes a limiting factor for use in runtime monitoring, because BAM requires monitoring to provide real-time information.

In this paper, HM-Hybrid solves such dilemma of batch update by resolving the correlations using memory resident structured indexes as a results cache. These indexes must be maintained consistently for all insert, delete, and update queries without accessing the base tables. This prevents access to the database except for executing batch updates.

Event correlation is an important idea for the recent event processing systems. And the event correlation is traditionally researched in the system management field. Hasan et al. [8] describes simple models for causal and temporal correlation. Because of the characteristics of diagnosis, the event correlation engines in this field usually do not mention about persistency for correlation state, in contrast to the monitoring systems for BAM.

Continuous queries [9] have been proposed for querying data streams. They essentially require the same processing mechanisms as used with monitors. They usually assume insertion only in contrast to HM-Hybrid, which allows all queries. This is due to differences in the available information and the constraints in processing queries.

There are many research reports on view maintenance in the field of materialized views. They include useful algorithms to maintain our results cache [10, 11, 12]. Gupta and Mumick categorized maintenance algorithms by the amounts of information that can be utilized when the views are refreshed.

Our model-driven approach can provide a complete set of access patterns by compilation as described in [4]. These access patterns, a full set of SQL statements, are equivalent to full information about the constraints for view maintenance. Hence our method can use almost full constraints, and the simplest and most effective maintenance functions can be derived from the full constraints.

In essence, our system works based on the constraint information. Thus it can be applied to any model and any rules from which such information can be extracted.

3. Hybrid event-processing architecture based on the model-driven approach

The basic concept of our hybrid architecture and its implementation HM-Hybrid are shown in this section. The model-driven approach which makes it efficient is also shown.

First, we describe the observation model underlying our model-driven approach. Then the model-driven result cache is illustrated. The model-driven result cache realizes the hybrid architecture. Finally, we describe how the hybrid architecture works.

3.1. Starting point: Observation model

To grasp business situations, a user defines an observation model that indicates which attribute of which event is to be used for the calculation of some KPI. One of the simplest and most typical KPI is "Turnaround-Time" for service orders, and its abstract observation model is shown in Figure 1.

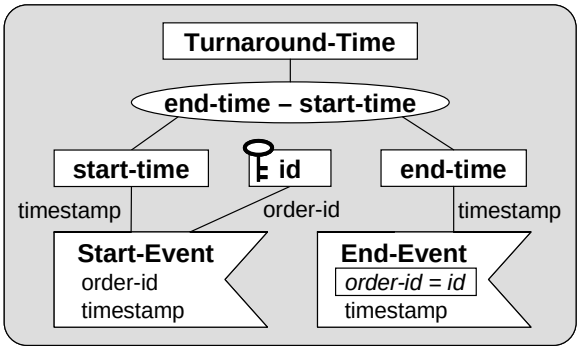

Figure 1. Observation model for "Turnaround-Time"

A "Start-Event" will be emitted when a customer buys a product, and an "End-Event" will be emitted when the enterprise ships the product. "Turnaround-Time" is calculated from these two events, which have at least two attributes, "order-id" and "timestamp". To calculate "Turnaround-Time" of an order, you can simply subtract the "timestamp" of a "Start-Event" from the "timestamp" of an "End-Event" when both events have the same "order-id". However, since the monitor system can never know when events make a pair, it saves the state as an object named "monitoring-context-instance" that has "order-id" as a primary-key and "timestamp" as an attribute when each "Start-Event" arrives. It correlates the state by querying the database with "order-id" when an "End-Event" arrives, calculates "Turnaround-Time", and saves the state again.

3.2. Generating result cache from the model

Figure 2 shows the concept of the model-driven approach for cache generation. The user first defines an observation model for monitoring. Since the given observation model describes all behaviors of the monitor application, we can extract all of its data access patterns by using the compilation approach. We can tell how many select or update statements can be used, and which columns of which tables might be changed.

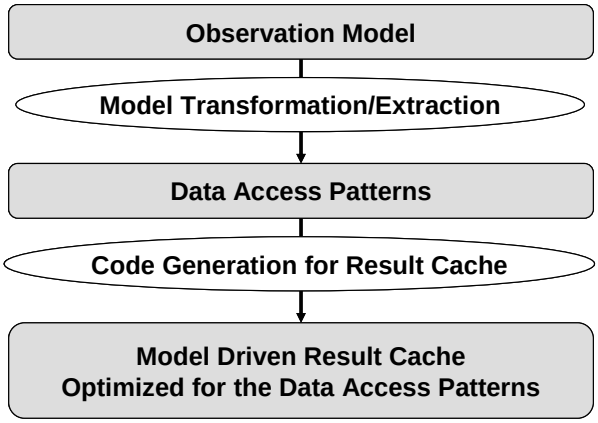

Figure 2. Model-driven cache generation

Once we have all of the data access patterns, we can generate a result cache which has optimized maintenance functions for the data access patterns. The data access patterns can be considered as constraints representing when and which target results in the cache are maintained. The maintenance functions are generated by using information about such constraints. In details, methods for deriving maintenance functions are well known in the field of materialized views [10, 11, 12].

In our approach, both the indexes for SQL select statements and the maintenance functions are generated as a result cache from the observation model. We call the generated result cache the "Model-driven result cache".

An index for an SQL select statement consists of a key and a condition (Figure 3). The key is from the parameterized predicates of a where-clause, and the key is used for the index entry. The value of the index entry will

be a result set of IDs for monitoring-context-instances. The condition is from the static predicates of the where-clause and the monitoring-context-instances must satisfy the condition when they are added to the index.

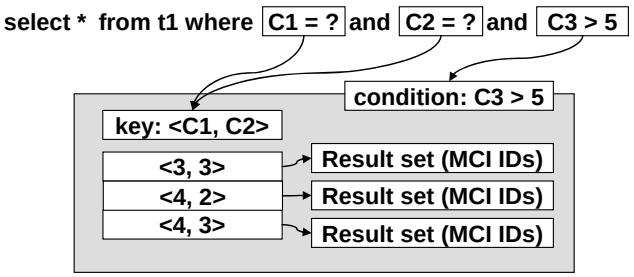

Figure 3. Generating an index from a where-clause

3.3. A new architecture: HM-Hybrid

HM-Hybrid consists of the model-driven result cache and a batch processor for the batch update function (Figure 4). For the monitor's requirement and real-time understanding of business situations, asynchronous batch updates are not usually allowed when the database should be responding to user queries.

In this hybrid architecture, the facade of the model-driven result cache substitutes for the database when users query with select statements, and this allows asynchronous batch update.

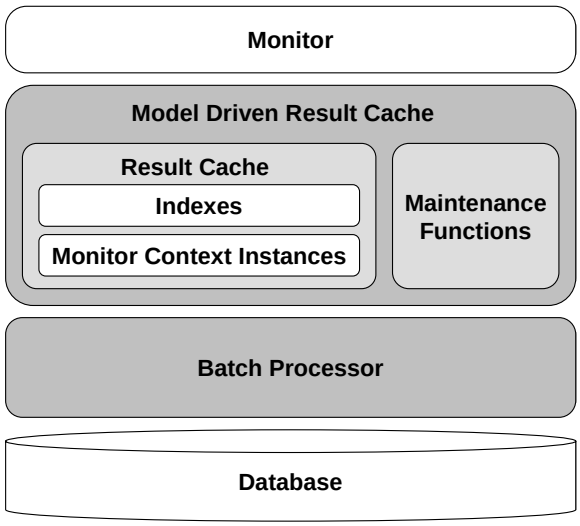

Figure 4. HM-Hybrid architecture for monitor

The model-driven result cache is the combination of the result cache and its maintenance functions. The result cache stores the search results of select statements as indexes which are linked to monitoring-context-instances. Maintenance functions are notified when the monitoring context instances are updated and this keeps the result cache fresh. When the monitoring-context-instances are updated, the updates are sent to the batch processor, and it executes periodic batch updates.

The detailed runtime architecture of HM-Hybrid is shown in Figure 5. HM-Hybrid consists of a model-driven result cache and a batch processor for batch updates.

When an event is emitted to the runtime, the event needs to be either correlated with existing monitoring context instances or creates a new monitoring-context-instance. For correlation, the correlation processor determines the index for the event from its event type and the key generation function corresponding to the index generates its key, and selects a result set from the index for that key. The result set includes monitoring-context-instance IDs of each monitoring-context-instance, and finally the event is correlated with these monitoring-context-instances.

The result set of the monitoring-context-instances passes to the instance processor. The instance processor calculates the KPIs from the correlated events and writes the updates back into the HM-Hybrid cache manager.

Then, the HM-Hybrid cache manager has two important roles. The first role is maintaining the result cache. The HM-Hybrid cache manager sends update requests to the maintenance functions. The maintenance functions evaluate the update requests with the target monitoring-context-instance and changes the corresponding indexes to keep them consistent. The second role is sending the update requests to the batch processor. The batch processor manages the updates as new record states, and consequently reduces the number of update queries. Because the multiple updates which updates the same record will be managed as a one state. It also periodically executes the batch update of the database.

3.4. Maintenance algorithm for result cache

The maintenance algorithm for the result cache is described in the pseudo code in Figure 6.

The code assumes that the result cache object *rc* provides ***add***, ***remove***, and ***contains*** methods as index manipulations. The derived three methods from observation model, ***createKey***, ***isKey***, and ***testCondition***, belong to *rc* as part of its maintenance functions.

First, the maintenance function creates two keys k_{old} and k_{new} from the corresponding values of the modified KPI. The k_{old} is used to remove the existing entry in the result cache, and k_{new} is used to add the new entry.

In Line 3, the function checks whether the result cache contains the target monitoring-context-instance *mci*. Then the derived condition is tested to decide if the updated *mci* should be indexed in the result cache, by considering the new value v_{new}.

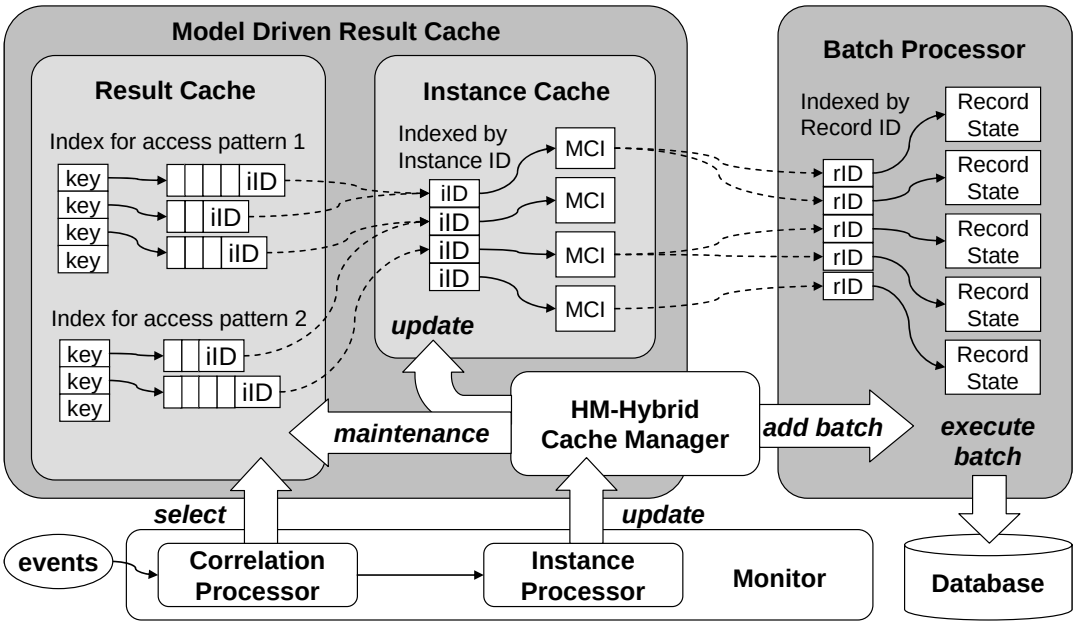

Figure 5. Event-processing flows and components of HM-Hybrid

Even if *mci* is still in the result cache, its entry may have been remapped to another key. Line 7 checks if the updated KPI was used to build the key of the index.

rc: the result cache object
mci: an updated monitoring-context-instance
m: the name of the updated KPI
v_{old}: the old value of the updated KPI
v_{new}: the new value of the updated KPI

```
1.  k_old = rc.createKey(mci, v_old)
2.  k_new = rc.createKey(mci, v_new)
3.  if (rc.contains(mci)) then
4.      if (! rc.testCondition(mci, v_new)) then
5.          rc.remove(k_old, mci)
6.      else
7.          if (rc.isKey(m)) then
8.              rc.remove(k_old, mci)
9.              rc.add(k_new, mci)
10.         endif
11.     endif
12. else
13.     if (rc.testCondition(mci, v_new)) then
14.         rc.add(k_new, mci)
15.     endif
16. endif
```

Figure 6. Maintenance algorithm

4. Preliminary evaluation

The testing environment, the scenario, and the results of our preliminary evaluation are described in this section. The results show that the HM-Hybrid monitor works six times faster than a non-hybrid one.

4.1. Evaluation method

We conducted performance tests for the evaluation of this hybrid architecture using a test scenario. We measured the performance by counting how many events could be processed. The event emitter driver on the application server emits as many events as the monitor system could process. A total of 17.5 select queries and 18.6 update queries were produced on average for each event. The batch update sends the update request which corresponds to an event as a unit to the database. The batch size defines how many units are sent together. To measure the effects of batch update, we varied the batch size from 1 to 800.

For comparison, we also conducted performance tests for a non-hybrid architecture (with a result cache only) using the same scenario. We chose to use the result cache alone for the comparison, although there are four possible combinations of result cache and batch update. This was because batch update alone is meaningless for monitoring, and the case of none is also meaningless since it is obviously inferior to the other cases in this scenario.

4.2. Testing environment

4.2.1. Testing platform. The testing platform consists of one 2.2 GHz dual core CPU machine and one 2.4 GHz dual core CPU machine connected via a 1 Gbps LAN (Figure 7). The first machine was the application server, and the second was the database server. The detailed specifications of each machine are shown below.

- **Application Server**
  - CPU: 2.2 GHz Dual core Opteron 275 x2, 2 MB 2L cache
  - Main Memory: 4 GB PC3200 RAM
  - Hard Disk: 80 GB SATA 7200 rpm
  - Operating System: Windows 2000 Service Pack 4
  - WebSphere Process Server 6.0.1.2

- **Database Server**
  - CPU: 2.4 GHz Opteron 250 x2, 1 MB 2L cache
  - Main Memory: 4 GB PC3200 RAM
  - Hard Disk: 73.4 GB 15 Krpm Ultra320 SCSI
  - Operating System: Windows 2000 Service Pack 4
  - DB2 UDB Enterprise Edition 8.1.8

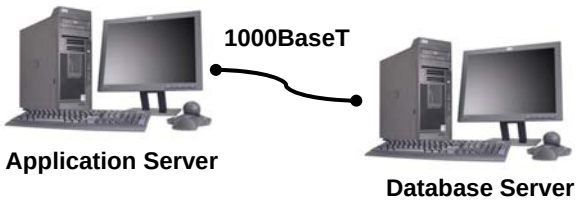

Figure 7. System Configuration

4.2.2. Testing scenario. The testing scenario is a business process to manage medical charts for patients (Figure 8). It uses 17.5 SQL select statements and 18.6 SQL insert, update, or delete statements for each event (on average).

In this scenario, a process instance was created for each visit of a patient. Seven activities and one conditional branch were included in the process model. Each process instance executes six activities while selecting one branch. A medical chart includes forty primitive type properties, and they are updated in every activity.

There are four types of events in this scenario, and two of them are generated during the execution of each activity. They are "Activity-Event", "Data-Changed-Event", "Process-Start-Event", and "Process-End-Event". An "Activity-Event" causes one select and three inserts to create a new sub-monitoring-context-instance. A "Data-Changed-Event" causes fifteen selects and fifteen updates to calculate the KPIs. A "Process-Start-Event" causes two inserts to create a new main monitoring-context-instance. A "Process end event" causes sixteen deletes for all of the monitoring-context-instances related to the process instance.

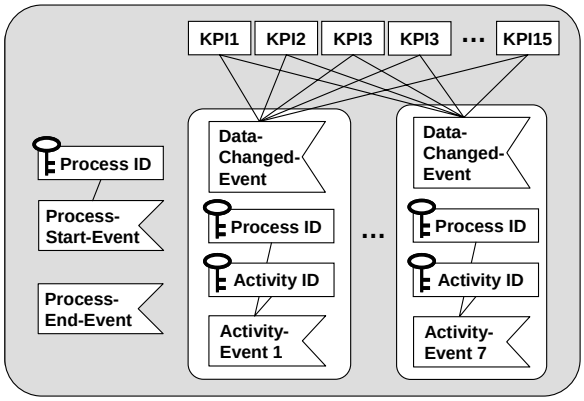

Figure 8. Observation model for testing scenario

4.3. Experimental results

Figure 9 shows the comparison results of the HM-Hybrid architecture and the non-hybrid architecture. The vertical axis represents the number of events processed each second. HM-Hybrid could process about six times as many events as the non-hybrid architecture.

The non-hybrid architecture reached its throughput peak around 200 events per second, because of the disk access bottleneck. HM-Hybrid could process 1207 events per second. It processed about 43000 queries totally in a second, since each event causes 36 queries. This incredible high throughput of processing queries is accomplished by two reasons.

The first reason is removing the disk access bottleneck by applying batch update. Batch update sends queries in a certain unit (250 queries in this case) and it makes the updates efficient, reducing the communication and maximize the disk utilization. Furthermore the actual number of update queries is reduced from the number of queries that the application sent, because the update queries which update a same record can be arranged into one update query. The second reason is the load balancing of the application server and database. Select queries are processed by the application server and update queries are processed by database server respectively on the hybrid architecture, while all select and updates queries are processed by the database server on the non-hybrid architecture.

HM-Hybrid achieved six times better performance, until the application server CPU became the new bottleneck.

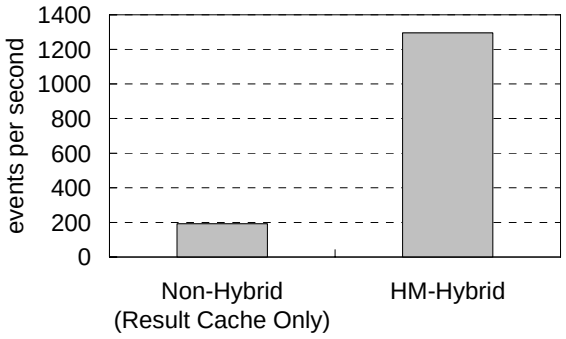

Figure 9. Comparison of throughput with and without batch update

Figure 10 shows a line graph of throughput (events per second) for each batch size with HM-Hybrid. The vertical axis represents the number of events which are processed in a second. The horizontal axis represents the batch size. The throughput curve reached its peak at a batch size of 250, and declined as it increased. This fell short of our expectations that the throughput curve of the simple JDBC client would also decline as the batch size increased rather than being almost satiated around the batch size of 200 (Figure 11). We believe the throughput degradation of HM-Hybrid above the batch size of 250 is due to a transition problem in the application server, and it needs to be investigated further.

However we reached almost the highest performance for the disk-based system, because the simple JDBC client was also satiated around a batch size 200.

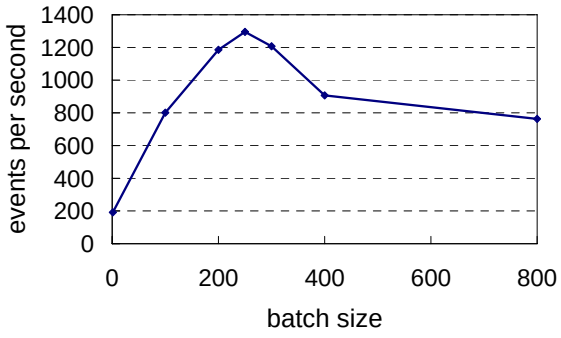

Figure 10. Throughput curve for batch size of HM-hybrid

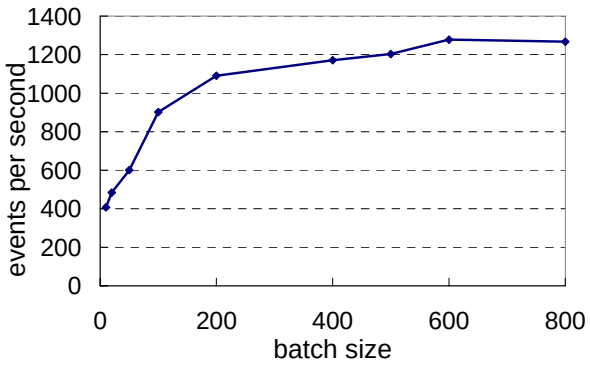

Figure 11. Throughput curve for batch size of simple JDBC client

5. Concluding remarks

In this paper, we introduced HM-Hybrid, a new monitor runtime architecture suitable for monitoring applications in which huge number of events occur. The combination of a result cache and batch update simultaneously supported high-performance event processing and persistence on the basis of a model-driven approach, using generated code for the result cache, which is optimized for data access patterns derived from an observation model.

This hybrid architecture allows use of a batch facility in a situation that is normally regarded as unsuitable. Therefore the architecture should be effective even when update accesses are numerous and select queries are also numerous. The hybrid architecture succeeded in processing about six times as many events as a naive cache design (by removing a disk access bottleneck). It will be the enhancement for monitoring in the dynamic and fast-changing business circumstances.

In the future, we will conduct a comparative analysis by constructing performance models of the architecture and other alternative architectures to removing the new CPU bottleneck.

References

[1] D.W. McCoy. "Business Activity Monitoring: Calm Before the Storm", Gartner Research, ID Number: LE-15-9727, 1st April, 2002.

[2] F. Leymann, D. Roller, and M.-T. Schmidt. "Web Services and Business Process Management", New Developments in Web Services and E-commercem, *IBM Systems Journal*, Vol. 41, No. 2, pp. 198-211, 2002.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. "Business Process

Execution Language for Web Services Specification Version 1.1", 5th May, 2003.
[4] L. Zeng, H. Lei, M. Dikun, H. Chang, and K. Bhaskaran. "Model-Driven Business Performance Management", *Proceedings of the 2005 IEEE International Conference on e-Business Engineering*, October, 2005.
[5] L. Baresi and S. Guinea. "Towards Dynamic Monitoring of WS-BPEL Processes", *Proceedings of the 3rd International Conference on Service Oriented Computing*, December, 2005.
[6] L. Baresi, C. Ghezzi, and S. Guinea. "Smart Monitors for Composed Services", *Proceedings of the 2nd International Conference on Service Oriented Computing*, November, 2004.
[7] K. Mahbub, and G. Spanoudakis. "Run-time Monitoring of Requirements for Systems Composed of Web-Services: Initial Implementation and Evaluation Experience", *Proceedings of the 2005 IEEE International Conference on Web Services*, July, 2005.
[8] M. Hasan, B. Sugla, and R. Viswanathan. "A Conceptual Framework for Network Management Event Correlation and Filtering Systems", *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, pp. 233-246, May, 1999.
[9] S. Babu and J. Widom. "Continuous Queries over Data Streams", *SIGMOD Record*, Vol. 30, No. 3, September, 2001.
[10] A. Gupta, and I.S. Mumick. "Maintenance of Materialized Views: Problems, Techniques, and Applications", *IEEE Data Engineering Bulletin*, Special Issue on Materialized Views and Data Warehousing, Vol. 18, No. 2, pp. 3-18, June, 1995.
[11] K.A. Ross, D. Srivastava, and S. Sudarshan. "Materialized View Maintenance and Integrity Constraint Checking: Trading Space and Time", *Proceedings of the ACM SIGMOD Conference*, pp. 447-458, June, 1996.
[12] S. Ceri, and J. Widom. "Deriving Production Rules for Incremental View Maintenance", *Proceedings of the 1991 VLDB Conference*, pp. 577-589, September, 1991.