Accelerating Sequence Event Detection through Condensed Composition

Yuanping Li [*1], Jun Wang [*2], Ling Feng [*3], Wenwei Xue [#4]

[*]*Department of Computer Science & Technology, Tsinghua University, Beijing, China*
{[1]liyp04@mails.,[2]wjun09@mails., [3]fengling@}tsinghua.edu.cn

[#]*Nokia Research Center, Beijing, China*
[4]wayne.xue@nokia.com

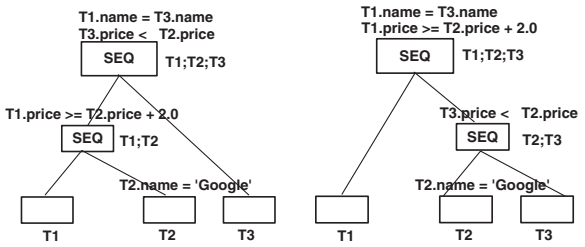
***Abstract*— Composite event processing systems are useful in various application domains such as stock data stream monitoring, RFID data management, web access pattern monitoring, etc. The tree-based method is a typical method for implementing composite event processing system. In this paper, we introduce an optimization method called "condensed composition" for the tree-based composite event processing. "Condensed composition" divides the events in groups to participate the composition. We analyze the reason why the performance is improved and utilizes the statistics of the "condensation degree" to dynamically decide using condensed composition or not for different data characteristics. In the experiments, the "condensed composition" method significantly improved the performance on the real stock data.**

Keywords – Composite event, context-aware data, streaming data, event detection, performance evaluation.

I. INTRODUCTION

Composite (Complex) event processing systems are useful in a variety of application domains such as stock price monitoring, RFID data management, web access pattern monitoring, etc. Such event detection is usually done on data coming as streams. Taking stock trading for example, a stock data stream has a schema with three attributes (*timestamp, name, price*). Each stock record in the stream can be viewed as a **primitive event**, denoted as e. For example, "(1),'IBM',6" states that IBM stock has a price 6 at timestamp (1). A **composite event** is a combination of several primitive events through the operators like sequence, conjunction, disjunction, negation, and Kleene closure. For example, $e_1; e_2; e_3$ is a composite event, comprised of three sequential primitive events e_1, e_2, and e_3. Suppose a fund manager may want to monitor a composite event on the stock data to buy or sell stocks. S/he may issue a query like Query 1 which detects a certain stock whose price is no less than Google's stock price plus 2.0 dollars and later is less than Google's stock price.

```
Query 1:
PATTERN T1;T2;T3
WHERE T1.name=T3.name
AND T2.name='Google'
AND T1.price >= T2.price + 2.0
AND T3.price < T2.price
WITHIN 500
RETURN T1, T2, T3
```



(a) Left-deep tree plan (b) Right-deep tree plan

Fig. 1. Two query trees for Query 1

In the "PATTERN" clause, T1,T2,T3 represent three **event classes**, the semicolon ";" denotes the sequential relationship of T1,T2, and T3. The predicates in the "WHERE" clause express the constraints over T1,T2,T3. In the "WITHIN" clause, a window size $w = 500$ is specified which requires $|T3.timestamp - T1.timestamp| <= w$. The "RETURN" clause specifies resulting composite events of T1;T2;T3 pattern which satisfy the query condition [1].

Tree-based method is a typical method for composite event detection. Two equivalent query trees for Query 1 are shown in Fig. 1. In a query tree, primitive events are fed into its leaf nodes, and the predicates are pushed down to the nodes of the trees as lower as possible. In this way, disqualified events can be filtered out earlier before they are transferred to the upper tree nodes. The two non-leaf tree nodes handle two sequential joins, which are T1;T2 and T1;T2;T3 in Fig. 1(a), and T2;T3 and T1;T2;T3 in Fig. 1(b).

A *batch-iterator model* is employed to process streaming data according to one of the query tree plans. If there are N composite events satisfying the query predicates, these predicates have to be evaluated at least N times. This bound is irrelevant to the change of tree structure. In addition, computations are also needed to exclude the unqualified primitive events by evaluating the predicates on them. Hence, the computations of predicate evaluations are more than N.

Considering continuous arrival of primitive events, and some events (primitive or composite) share values for the same attributes, we ask a question: *can we eliminate some predicate*

[1]The notations adopted are from [10]

978-1-4244-8812-4/10/$26.00 ©2010 IEEE

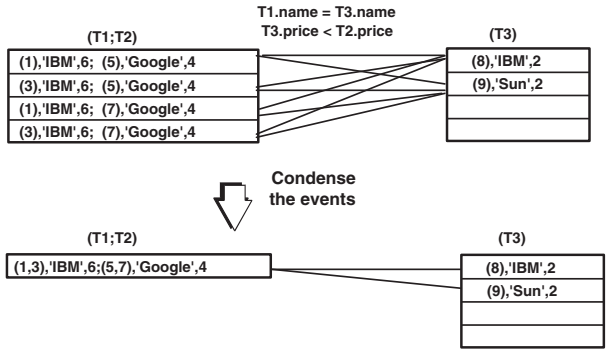


Fig. 2. Nest loop join for sequential join

evaluations upon event attributes for a faster sequence pattern detection? To illustrate, let's see Fig. 2, which corresponds to the left and right children of the root node in Fig. 1(a). The connecting lines stand for primitive (or composite) event combinations that need predicate evaluations. Originally, 8 combinations are needed to participate the predicate evaluations (T1.name=T3.name and T3.price<T2.price). By condensing the events according to the attribute values, only 2 combinations are needed.

Based on this observation, we present a condensed composition method to reduce the computations for predicate evaluation. Multiple primitive (or composite) events are grouped together according to the predicates and take part in join predicate evaluation as one condensed event. In this way, the number of predicate evaluations is reduced. Besides, by measuring the condensation degree in the event buffers, we provide a way to switch event condensation on or off accordingly.

In this study, we assume that primitive events in streaming data are ordered by timestamp and the tree plan has already been optimized by tree structure optimization method [10]. The rest of the paper is organized as follows. In Section 2, we describe some closely related work. We present our condensed composition method in Section 3. In Section 4, we analyze the complexity of the tree-based method and utilize a statistical "condensation degree" measurement to dynamically choose whether to use condensed composition for queries. In Section 5, we test the performance of the approach on both synthetic and real stock data. We conclude the paper and discuss future work in Section 6.

II. Related Work

Complex event detection model. There exist typically two ways for composite event detection. One is the NFA-based method which uses nondeterministic finite automata (NFA) or its variants [1], [4], [5], [15]. Wu *et al.* built a high-performance composite event detection system over streams using NFA model [15]. They optimized system performance by using active instance stacks, pushing predicates down to SSC (sequence scan and construction) and pushing windows down. Agrawal *et al.* explored the use of shared match buffers and merging equivalent multiple runs to improve the performance [1]. Cao *et al.* incorporated inference techniques into event detection for distributed RFID tracking and monitoring, and used writable onboard tag storage for transferring queries [2]. Compared with the NFA-based method, the tree-based method was proposed in [10], [11], which allows dynamic tree structure optimization based on a certain cost model. Such an optimization makes the tree-based method more efficient than the NFA-based method.

Indexing in complex event detection. Demers *et al.* built an NFA-based event detection system called Cayuge [4], employing two types of indexes on predicates to improve the NFA performance [4]. One type of indexes is called static index focusing on the comparison between an attribute's value and a constant. The other type of indexes is called dynamic index, focusing on the comparison between two attributes' values. Cayuage implements the indexes of equality comparison of two attributes' values. Index schemes have also been designed for selecting data and queries in streams (e.g., on geo-spatial streaming data [12], [13], [7]). Although not much work was reported on the index-based join algorithms for the tree-based detection method, such algorithms have the potential in improving performance for certain types of predicates. Compared with indexing methods that focus on the index structures, our method focuses on the organization of the intermediate resulting events. Our method is more like merging equivalent runs [1] for the NFA-based detection model. But our data structures are different and have more adaptiveness for the tree-based detection model which can utilize dynamic tree structure optimization.

Stream processing. Previous stream processing systems like TelegraphCQ [8], [3] were able to share operators among different continuous queries on streams. Regular expression matching systems, which is mainly used in string stream processing, could efficiently process the queries with regular expressiveness [9]. Compared with general data stream management systems, event detection systems have distinct class of queries [4] and uses specific execution and optimization strategies for these queries. In terms of regular expression matching, complex event detection systems have languages with richer expressiveness [1].

Other optimizations for sequential pattern detection To speed-up sequential pattern detection, Sadri *et al.* developed a generalized KMP algorithm to minimize repeated passes over the same data for sequence queries, when a query pattern has interdependent elements [14].

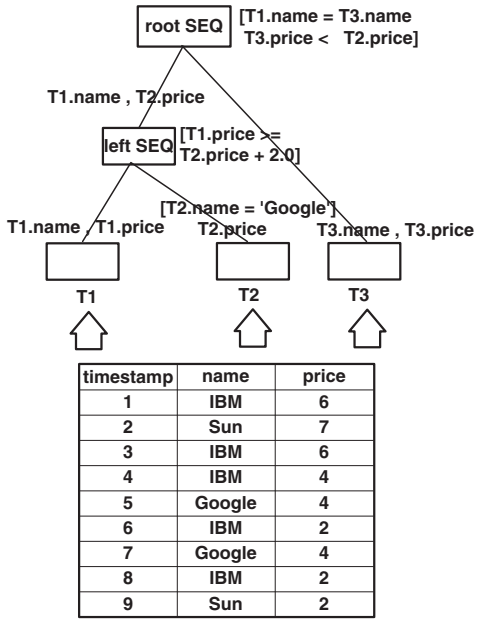
III. Condensed Composition

In this section, we describe our condensed event composition to reduce the computations of predicate evaluation.

A. Condensed Composition

The condensed composition technique contains the following two steps.

1) **Preprocessing.** Before setting up condensed composition data structures, we need to select condensation terms according to the predicates. A ***condensation term*** is an (event class name, attribute name) pair which is denoted by **event-class.attribute**.



timestamp	name	price
1	IBM	6
2	Sun	7
3	IBM	6
4	IBM	4
5	Google	4
6	IBM	2
7	Google	4
8	IBM	2
9	Sun	2

Fig. 3. The condensation terms for Query 1

When the tree plan is generated, we add condensation terms onto each tree node TN iff

- The term **event-class.attribute** is in one of the predicates of the tree nodes upper than TN.
- The event class of **event-class.attribute** is in the leaf nodes of the sub tree whose root is TN .

For Query 1, the tree plan after the condensation terms are added, is shown in Fig. 3. T1.name, T2.price are the condensation terms for the left SEQ tree node. {T1.name, T1.price} are the condensation terms for the leaf node T1. {T2.price} is the condensation term for the leaf node T2. {T3.name, T3.price} are the condensation terms for the leaf node T3. We provide some stock data examples in a stream and put them to the leaf nodes which represent event classes. But, note that this does not mean the tree-based method handles only one schema. Different event classes can have different schemas. The preprocessing is similar to that for π optimization in relational database [6], but its usage is different which will be introduced in the following.

2) **Setting up data structures.** In the previous systems, a composite event is implemented as a list of primitive event. In condensed composition, there are two types of ***condensed event***, condensed composite event and condensed primitive event. A ***condensed composite event*** is a list of condensed primitive event. A ***condensed primitive event*** is constituted by a set of primitive events. The primitive events are organized into a condensed primitive event when they have the same values for the condensation terms of the tree node. Each tree node has an event buffer to maintain the intermediate processing results.

For example, let the batch size = 1 in Query 1, i.e., the system takes 1 primitive event from the stream, processes it and then takes the next primitive event.

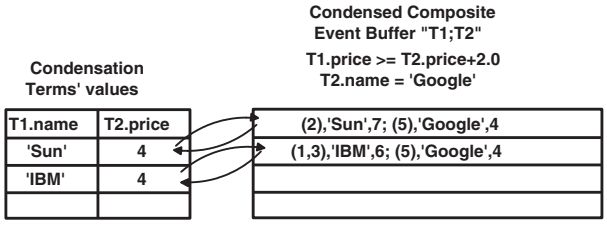


Fig. 4. The event buffer of the left SEQ node after the 5th primitive event is processed

- When the 1st event has been processed, the T1's event buffer has one event "(1),'IBM',6".
- When the 2nd event has been processed, the T1's event buffer has two events "(1),'IBM',6" and "(2),'Sun',7".
- When the 3rd event has been processed, since the T1's condensation terms are T1.name and T1.price. The 3rd event has the same T1.name and T1.price values, "(1), 'IBM', 6" and "(3), 'IBM', 6" are condensed into one event "(1,3),'IBM',6" and it is added to the end of the buffer. The T1's event buffer has two events "(2),'Sun',7" and "(1,3),'IBM',6".
- When the 4th event has been processed, the T1's event buffer has three events "(2),'Sun',7", " (1,3),'IBM',6" and "(4),'IBM',4".
- When the 5th event has been processed, the T2's predicate "T2.name='Google'" is satisfied by the 5th event. The T1 and T2 are joined at the left SEQ tree node. The event buffer of the left SEQ tree node has two condensed composite events "(2),'Sun',7;(5),'Google',4" and "(1,3),'IBM',6;(5),'Google',4", as shown in Fig. 4.

B. Event Insertion

When an event **e** is inserted into the condensed event buffer, **e**'s condensation terms are evaluated. Through the condensation terms, **e** will be condensed to the existing event which has the same condensation term values. If no event in the event buffer has the same condensation term values with **e**, **e** will be inserted to the end of the event buffer, and **e**'s condensation term values will be added to the condensation term value list. For example, in Fig. 4, when a new event "(2),'Sun',7;(7),'Google',4" is to be inserted into the event buffer, we first look for T1.name='Sun' and T2.price=4 in the condensation term value list, then through the pointer from the condensation term value list to the condensed composite event buffer, "(2),'Sun',7; (7),'Google',4" and "(2),'Sun',7;(5),'Google',4" are condensed to one condensed event "(2),'Sun',7;(5,7),'Google',4". The condensed event is added to the end of the event buffer. The result is in Fig. 5.

One may argue that if the stream attribute values are real numbers but not integers, the probabilities, that there are many tuples in a small time window w will have the same values for one or more attributes, are low. But actually, the benefit

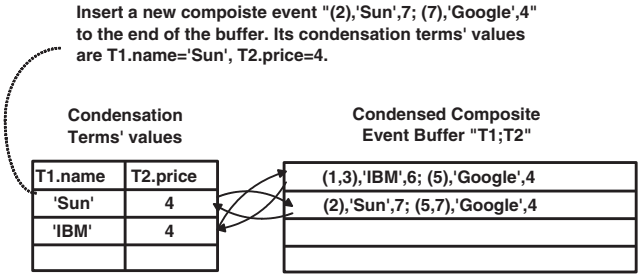


Fig. 5. Insert one event into the event buffer of the left SEQ tree node for Query 1

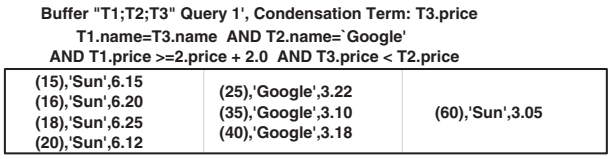


Fig. 6. A condensed composite event example

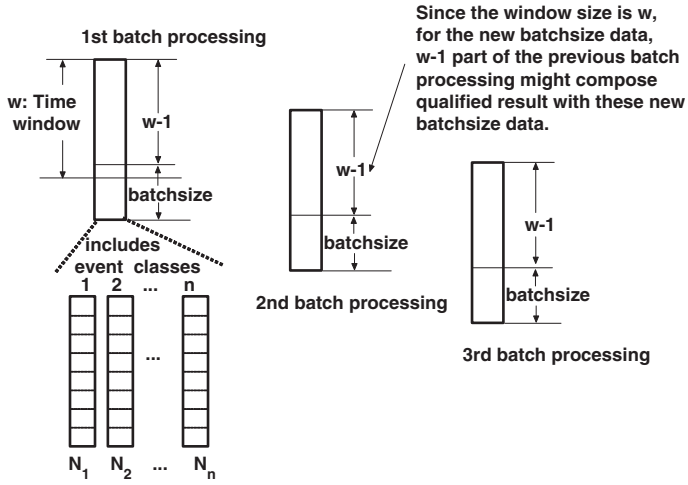


Fig. 7. Analysis of the computational complexity in the batch-iterator model

of condensed composition is also on interval style composite events, as explained in the following.

Consider Query 1', in which T4 is newly added compared with Query 1. "T3.price $>$ T4.price" is a new predicate. Suppose the optimal tree structure is a left-deep tree according to the data characteristics.

```
Query 1':
PATTERN T1;T2;T3;T4
WHERE T1.name=T3.name
AND T2.name='Google'
AND T1.price >= T2.price + 2.0
AND T3.price < T2.price
AND T3.price > T4.price
WITHIN 500
RETURN T1, T2, T3, T4
```

The prices are real numbers. As shown in Fig. 6, the condensation term is T3.price for the buffer of the "T1;T2;T3" node. When condensed composition is not used, the event buffer "T1;T2;T3" has the events "(15); (25); (60)", "(15); (35); (60)", ... , "(20); (25); (60)", "(20); (35); (60)", "(20); (40); (60)". Here, we use only timestamps to denote each primitive event for space limitations. We get one condensed composite event "(15,16,18,20); (25,35,40); (60)" because the events before condensation have the same value of T3.price. All combinations of the three time interval (15,16,18,20) of T1, (25,35,40) of T2 and (60) of T3 satisfy the sequential constraint and the predicates which involve T1, T2 and T3. (But note that they are not rigorously continuous time interval.) Here, we use only timestamps to denote each primitive event for brevity.

C. Output Generation

When the data structure is set up, the output we get is a condensed composite event, i.e., a list of condensed primitive events. Directly outputting the condensed composite event by using every combinations of the primitive event from the condensed set is incorrect, because not all of the combinations satisfy the predicates in the query. Hence, we add a pointer chain structure behind the condensed composite event to get the correct output. Each primitive event **e**, in one condensed primitive event, obtains a pointer to a set of primitive events as **e**'s successors. This is done in conjunction with the event composition. The point chains finally facilitate outputting the results correctly.

IV. Complexity Analysis

From the basic algorithms introduced in [10], all the operators constitute the base of composite event computations. We can see that the computational complexity of Kleene closure and disjunction is relative low, i.e. linear with the input data cardinality. Negation operator also incorporates an operation of sequence. The main computational cost is from the join operators, such as sequence and conjunction operators. Optimization on the high-complexity operators will have a fundamental influence on the entire event processing system, no matter whether it is a centralized, distributed or parallel system.

Suppose the predicates are pushed down onto each tree node. The data are processed in the batch-iterator model in which the streaming data are processed in batches [10]. Consider the worst case where the selectivity of predicates, including the time predicates, is 1. Let w be the time window size and the number of primitive events with the same timestamp be 1. After one batch processing, $w-1$ data remain for the next batch processing because these $w-1$ data can also combine with some new data, without breaking the window size constraint. Without loss of generality, we suppose $batchsize < w-1$ and the cost of predicates evaluation for one outputted event is 1. We put one stream into different event classes' buffers. Then the computations for predicate evaluation in the event composition per second is

$$C = \frac{(batchsize + (w-1))^n - (w-1)^n}{batchsize/rate}$$

where

n: the number of event classes;

$rate$: the rate (events/second) of event arrivals;

$batchsize$: the size of one batch of data that be input to the tree plan.

In other words, in the best previous work, at least C is needed, no matter whether the tree plan structure selection method is used. In the case of a pattern with only sequence operators, let the selectivity of the time predicates be P_t. For any two event e_1,e_2 of different event classes, if the probability $Pr(e_1.timestamp < e_2.timestamp) = 1/2$, then we can get an estimate of P_t, $(1/2)^{n-1}$. The computations needed in the event composition per second is

$$C_{seq} = P_t \times C$$
$$= (1/2)^{n-1} \times \frac{(batchsize + (w-1))^n - (w-1)^n}{batchsize/rate}$$

The condensed composition does not necessarily fulfill the computation C for predicate evaluations. This is because the event buffer is organized in condensed events. When two condensed event are composed, the predicates need to be evaluated only once.

However, it is noteworthy that whether the condensed composition method can improve the performance or not depends on the condensation degree obtained on the data. In the implementation, we can measure the condensation degree d for the event buffer of each tree node by

$$d = \frac{size_1}{size_2} \leq 1$$

where

$size_1$: the number of events after condensation;

$size_2$: the number of events before condensation.

For a sequence operator node, the benefit of condensed composition is

$$benefit = a \times (size_L \times size_L - d_L \times d_R \times size_L \times size_R)$$

The cost incurred by the new data structure and operations is

$$cost = b \times (size_L + size_R)$$

where a and b are coefficients for specific implementations, $size_L$ is the number of events in the left child before condensation, $size_R$ is the number of events in the right child before condensation, d_L, d_R are the condensation degrees of the left child and the right child. We can set a threshold for $benefit - cost$ and the system can automatically switch condensed composition on/off for the buffer.

Correctness. The result remains the same as the non condensed composition. The events are grouped in the left child and the right child of the sequence operator, the predicate evaluation results hold for all events in a group due to the same condensation terms' values. Hence, all possible combinations of primitive events will be evaluated. In addition, the outputted events are evaluated by all the predicates, so all the outputted events satisfy the query constraints.

Space Complexity. In the respect of space complexity, condense composition needs new pointers to group the events. In the worst case, these added space cost is *O(intermediate-event-num)*. *intermediate-event-num* is the number of intermediate events when the condense composition is not used.

In summary, we analyze the complexity of sequential event composition in this section. The computational complexity of predicate evaluation can be reduced when condensation degree is high.

V. Performance Evaluation

We implemented condensed composition in a Java-based prototype system. The computer to execute the prototype system runs Windows 7 , and Java virtual machine 1.6.0_16, and it has a Intel Core2 Quad Q9400 2.66GHz CPU, 2GB RAM. In each experiment, the program is run 30 times and the average of the measurements is used. Since we want good response time for each event, we set batch size to be 1, which means events do not accumulate before being processed. The performance is mainly measured by the system ***throughput*** that is the maximum number of incoming events the system can handle per second.

A. Experiments on Synthetic Data

The first experiment is for Query 2 and is based on the optimal tree structure after optimization. The reason for designing such a query is we can analyze the influence of window size and class number through varying them. Data of n event classes are synthesized. Their prices are n-dimensional vector $\mathbf{x}$. The stock price at time t is

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \delta$$

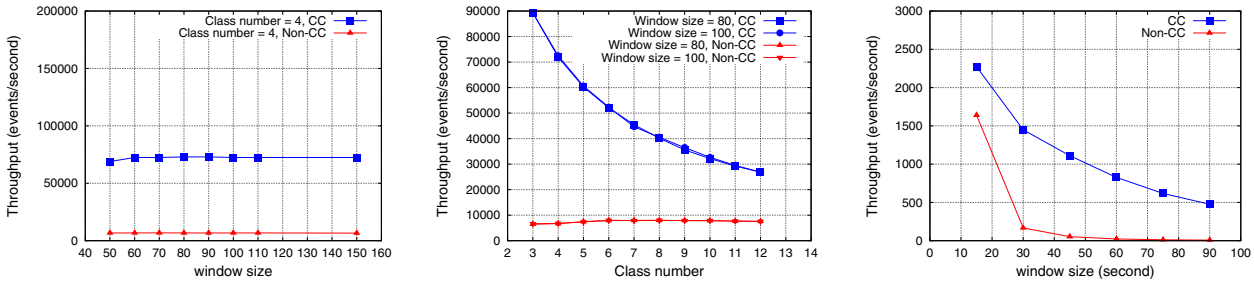
For one stock X, X_0 is generated randomly, from a uniform distribution on [0.00, 100.00). Each element in δ is randomly selected from a uniform distribution on [-5.00, 5.00). Through this random walk data generation method, if $X_t > 100.00$, we bound $X_t = 100.00$ and if $X_t < 0.00$, we bound X_t = 0.00.

```
Query 2:
PATTERN T1;T2;T3;...;Tn
WHERE T1.price>T2.price AND T1.price>T3.price
...
AND T1.price>Tn.price AND T1.name='name1'
AND T2.name='name2'a AND T3.name='name3'
...
AND Tn.name='namen'
WITHIN windowsize
RETURN T1,T2,T3,...,Tn
```

The experiment results are as follows.

Window size. The performance comparison with condensed composition (CC) and non condensed composition is shown in Fig. 8(a). We can see that the throughput of CC outperforms that of Non-CC in the synthetic data.

Class number. When the class number varies, we get the throughput measurements shown in Fig. 8(b). The throughput of CC decreases when the class number increases. The throughput of CC outperforms that of Non-CC.



(a) The throughput when the window size varies, class number is 4 (b) The throughput when the class number varies (c) The performance on the real stock data

Fig. 8. Performance evaluation

B. Experiments on Real Stock Data

We further conducted an experiment on the real stock data. The first 800 companies stocks were collected from the Shang Hai stock exchange market in the time from 2010-03-09 10:48:28 to 2010-03-09 11:20:13 [2]. The stock data was published every 5 seconds. Query 3 is a query on the real data. It is to detect a stock whose price is lower than the price of the stock '600210' at the beginning and is higher than the price of the stock '600210' later.

```
Query 3:
PATTERN T1;T2;T3;T4
WHERE T1.code=T3.code AND T2.code=T4.code
AND T2.code='600210'
AND T1.price < T2.price-0.02
AND T3.price > T4.price+0.02
WITHIN windowsize
RETURN T1,T2,T3,T4
```

The experimental results are shown in Fig. 8(c). We can see that on the real data, the condensed composition can also improve the system performance greatly. When the window size increases, the benefit of condensed composition is evident. When the window size is 90 seconds, the throughput of using condensed composition is about 100 times of the throughput of not using condensed composition.

VI. Conclusion

High performance composite event processing is required in the real-time applications. In this paper, we propose a condensed composition method for the composite event detection. Such a method can significantly improve the system performance when condensation degree is high. One future work might be to generate condensation terms based on the predicates of only the current tree node by organizing condensed composite events directly with composite events, not with a list of condensed primitive events.

VII. Acknowledgements

The work is supported by National Natural Science Foundation of China (60773156), Chinese National 863 Advanced Technology Program (2008AA01Z132), Research Fund for the Doctoral Program of Chinese Higher Education (20070003089), Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry. We also thank the support from Nokia Research Center in Beijing.

[2]http://www.cnblogs.com/raymond19840709/archive/2008/07/31/1257048.html

References

[1] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *Proceedings of SIGMOD*, pages 147–160. ACM, 2008.

[2] Z. Cao, Y. Diao, and P. Shenoy. Architectural considerations for distributed rfid tracking and monitoring. In *the 5th International Workshop on Networking Meets Databases (NetDB)*, 2009.

[3] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, F. Reiss, and M. Shah. TelegraphCQ: continuous dataflow processing. In *Proceedings of SIGMOD*, pages 668–668. ACM, 2003.

[4] A. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. White. Cayuga: A general purpose event monitoring system. In *Proceedings of CIDR*, pages 412–422.

[5] F. Fabret, H. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. *ACM SIGMOD Record*, 30(2):115–126, 2001.

[6] H. Garcia-Molina, J. Ullman, and J. Widom. *Database system implementation*. Prentice Hall, 2000.

[7] Q. Hart and M. Gertz. Indexing query regions for streaming geospatial data. In *2nd Workshop on Spatio-temporal Database Management, STDBM04*, 2004.

[8] S. Madden, M. Shah, J. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *Proceedings of SIGMOD*, pages 49–60, 2002.

[9] A. Majumder, R. Rastogi, and S. Vanama. Scalable regular expression matching on data streams. In *Proceedings of SIGMOD*, pages 161–172, 2008.

[10] Y. Mei and S. Madden. Zstream: A cost-based query processor for adaptively detecting composite events. In *Proceedings SIGMOD*, pages 193–206. ACM, 2009.

[11] J. Mikkelsen. Efficient complex event processing over data streams. Master's thesis, University of Southern Denmark, 2009.

[12] J. Min. A Query Index for Stream Data Using Interval Skip Lists Exploiting Locality. In *ICCS 2007, Part 1, LNCS 4487*, pages 245–252, 2007.

[13] J. Park, B. Hong, and C. Ban. A continuous query index for processing queries on rfid data stream. In *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007*, pages 138–145, 2007.

[14] R. Sadri, C. Zaniolo, A. Zarkesh, and J. Adibi. Expressing and optimizing sequence queries in database systems. *ACM Transactions on Database Systems*, 29(2):282–318, 2004.

[15] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *Proceedings of SIGMOD*, pages 407–418, 2006.