APAM: Adaptive Eager-Lazy Hybrid Evaluation of Event Patterns for Low Latency

Ilyeop Yi[†] Jae-Gil Lee[‡*] Kyu-Young Whang[†*]
[†]School of Computing, KAIST, Daejeon, Korea
[‡]Graduate School of Knowledge Service Engineering, KAIST, Daejeon, Korea
{iyyi, kywhang}@mozart.kaist.ac.kr jaegil@kaist.ac.kr

ABSTRACT

Event pattern detection refers to identifying combinations of events matched to a user-specified query event pattern from a real-time event stream. *Latency* is an important measure of the performance of an event pattern detection system. Existing methods can be classified into the eager evaluation method and the lazy evaluation method depending on when each event arrival is evaluated. These methods have advantages and disadvantages in terms of latency depending on the event arrival rate. In this paper, we propose a *hybrid eager-lazy* evaluation method that combines the advantages of both methods. For each event type, the hybrid method, which we call *APAM* (*Adaptive Partitioning-And-Merging*), determines which method to use: eager or lazy. We also propose a formal cost model to estimate the latency and propose a method of finding the optimal partition based on the cost model. Finally, we show through experiments that our method can improve the latency by up to 361.48 times over the eager evaluation method and 27.94 times over the lazy evaluation method using a synthetic data set.

1. INTRODUCTION

Event pattern detection refers to finding combinations of events (called *pattern instances*) matched to a user-specified event pattern from a real-time event stream [8]. Throughput and latency are the major measures of the performance of event pattern detection. *Throughput* refers to the number of events processed per unit time [11], and *latency* the time elapsed between the arrival of the last event of a pattern instance matched to a query event pattern and the point at which the system identifies that the pattern instance is matched [12]. While both measures are important, latency is highly significant for mission-critical or time-critical applications. However, most existing studies assume that the event arrival rate is extremely fast, and thus, only focus on improving the throughput [1, 2, 4, 5, 10, 11, 13, 14].

Existing event pattern detection methods can be classified into the eager evaluation method and the lazy evaluation method depending on when each arriving event is evaluated. Each method has its advantages and disadvantages in terms of latency depending on the event arrival rate.

- The *eager evaluation method* evaluates all events immediately when they arrive. Thus, events are always evaluated in the order in which they arrive. An advantage of the eager evaluation method is that latency is low when the event arrival rate is slow. The slower the event arrival rate is, the more idle time there will be. The eager evaluation method utilizes this idle time to evaluate arriving events immediately. Because most events of a pattern instance have been detected prior to the arrival of the last event of that pattern instance, the eager evaluation method can detect the pattern instance as soon as the last event arrives. However, a disadvantage of the eager evaluation method is that latency may be high when the event arrival rate is fast. The faster the event arrival rate is, the less idle time there will be. Hence, events may *not* be evaluated immediately when they arrive, and these evaluations may be delayed. In this case, latency is significantly affected (i.e., gets higher) by lower throughput, which decreases because the order in which events are evaluated may not be optimal in eager evaluation.
- The *lazy evaluation method* defers evaluations of arriving events and evaluate them in batch later. Thus, events can be evaluated in any order. An advantage of the lazy evaluation method is that latency is low when the event arrival rate is fast. As mentioned above, when the event arrival rate is fast, the latency is heavily dependent on the throughput. Given that events can always be evaluated in the optimal order, the throughput is high and latency is low. However, a disadvantage of the lazy evaluation method is that the latency is high when the event arrival rate is slow. Even when there is sufficient idle time, the lazy evaluation method defers all evaluations of arriving events until the batch evaluation starts. When the last event of a pattern instance arrives, all the events that arrived prior to the last event must be evaluated in batch. Hence, the time taken to detect the pattern instance in this case is delayed.

In this paper, we propose a *hybrid eager-lazy evaluation method*, which combines the advantages of both methods providing low latency at any event arrival rate. For each event type, the proposed method determines which method to use: eager or lazy. At slower event arrival rate, more event types are evaluated by the eager evaluation method in order to fully utilize the idle time. At faster event arrival rate, more event types are evaluated by the lazy evaluation method in order to evaluate more events in the optimal order.

Toward this goal, we first propose a hybrid method **APAM** (meaning *Adaptive Partitioning-And-Merging*). **APAM** partitions event types into an eager evaluation group and a lazy evaluation group, and merges the results of evaluations of the eager evaluation group with the deferred events of the lazy evaluation group to construct

*Jae-Gil Lee and Kyu-Young Whang are the co-corresponding authors

CIKM'16 , October 24-28, 2016, Indianapolis, IN, USA

DOI: http://dx.doi.org/10.1145/2983323.2983680

the pattern instances. Second, we propose a method of selecting the optimal plan. **APAM** is able to use various plans according to how event types are partitioned. The proposed method estimates the latency of each plan and selects the plan with the lowest latency. We define a cost model to estimate the latency. Finally, we empirically validate the performance of our optimal plan selection method and show that **APAM** achieves the lower-bound of latency of both existing methods at any event arrival rate.

2. PRELIMINARIES

An *event* is a tuple that represents an instantaneous and atomic occurrence of interest at a point in time [3, 8]. Each event has a set of attributes according to the schema defined for the type of that event. Essentially, an event has the name of the type of that event and the time of the event associated with (when the event occurred) as attributes. For ease of explanation, an event type is denoted by an upper-case letter (e.g., A), and an event by a lower-case letter with its timestamp value subscripted (e.g., a_1). *Event stream* is a linearly ordered sequence of events [8, 11] and is the input to the event pattern detection system. An event stream may contain events of heterogeneous types. In the event stream, events are assumed to be sorted by their timestamp values.

The query language of an event pattern detection system specifies the event pattern to be detected. We use a declarative query language [1, 10, 11, 14]. A query event pattern consists of the following three clauses:

PATTERN	*Event Pattern*
WHERE	*Predicates on the Event Attributes*
WITHIN	*Time Window*

A PATTERN clause specifies an event pattern by using event types and event operators that represent the correlations among event types. A sequence operator (;) is the primary event operator focused on by the event pattern detection area [7, 10, 11], and thus, we concentrate on the event patterns specified by sequence operators. $A; B$ indicates that an event of type B occurs after an event of type A has occurred. All the combinations of events (a_x, b_y), where $x < y$, are matched to the event pattern $A; B$.

WHERE and WITHIN clauses specify the predicate and the time conditions that the events matched to the event pattern satisfy. A WHERE clause specifies the predicates on event attributes. There are two types of predicates: a simple and a parameterized predicate. A simple predicate is a predicate on an attribute of one event type (e.g., $A.attr1 > 10$). A parameterized predicate is a predicate between attributes of different event types (e.g., $A.attr1 = B.attr1$). A WITHIN clause specifies a time window that represents the time period within which events matched must occur.

3. OVERVIEW OF APAM

In **APAM**, event types are partitioned into two groups: the *eager evaluation group (EEG)* and the *lazy evaluation group (LEG)*. Events in the *EEG* are evaluated immediately when they arrive, and evaluations of events in the *LEG* are deferred until the last event of a pattern instance, called the *final event* [10], arrives. Whenever the final event arrives, events whose evaluations are deferred and the results of the eager evaluations are merged to construct pattern instances. Parameterized predicates are also partitioned into two groups along with the event types. Predicates involving the event types of the *EEG* are grouped into the *EEG*, and other predicates (those involving the event types of the *LEG* and those involving the event types of both groups) are grouped into the *LEG*. Predicates are evaluated when events of the same group are evaluated.

Figure 1 shows the architecture of an event pattern detection system that uses **APAM**. For simplicity, we assume that events whose types are not specified in the query event pattern and who do not satisfy the simple predicate are filtered out beforehand. As shown in the figure, the system consists of four modules.

- The *Partitioner* determines the evaluation method of each arriving event according to the plan selected by the *Plan Optimizer*. The *Plan Optimizer* selects the plan that minimizes the latency according to the event arrival rate, which will be explained in Section 4. In addition, it maintains the statistics needed to select the optimal plan. Statistics are calculated using simple windowed averages based on sampling [10].
- The *Eager Evaluator* evaluates the *EEG* events immediately. As a result, pattern instances matched to the partial query event pattern consisting of the event types in the *EEG* are constructed and stored in the eager evaluation result buffer.
- The *Lazy Evaluator* stores the *LEG* events into deferred event buffers allocated per event type. Evaluations of these events are deferred until the final event arrives.
- The *Merger* constructs pattern instances matched to the query event pattern using the results of eager evaluations and the events whose evaluations are deferred. In Section 5, we explain merging in more detail.

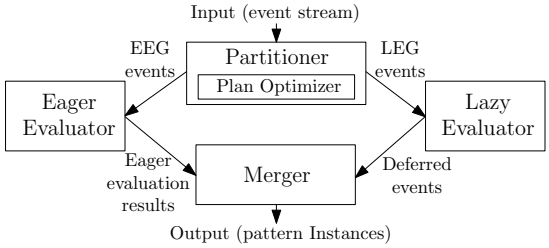


Figure 1: The architecture of APAM.

4. OPTIMAL PLAN SELECTION

In this section, we explain the optimal plan selection method reducing the latency of **APAM**. Before proceeding, we summarize the notation used in the following sections in Table 1. For an event pattern whose length is l, the number of possible plans is $\sum_{i=0}^{l} \binom{l}{i}$. To reduce the search space, we fix the matching order to the optimal matching order. A *matching order* is an order by which the events matched are detected, and the *optimal matching order* corresponds to the optimal join order [6]. Here, we use a heuristic method to find the optimal order processing more selective matching earlier. Thus, we consider $l+1$ evaluation plans resulting from binary partitioning of event types sorted by the optimal matching order as candidate plans. We then estimate the latency of each candidate plan. The candidate plan with the lowest estimated latency is selected as the optimal plan. For example, $/BCAD$, B/CAD, BC/AD, ABC/D, and $ABCD/$ are the candidate plans of $A; B; C; D$, whose optimal matching order is B, C, A, D. Here, BC/AD represents the evaluation plan where B and C are grouped into the *EEG*, A and D are grouped into the *LEG*, and the matching order is B, C, A, D. Notice that the plan resulting from binary partitioning where three event types are grouped into the *EEG* is ABC/D rather than BCA/D. This is because the matching order of the *EEG* is fixed to the order specified in the query event pattern.

4.1 Queueing model

To estimate the latency of the evaluation plan, we model **APAM** using an M/G/1 queue. Characteristics of the M/G/1 queue is summarized as follows [9]:

- *M*: Events arrive at the input queue of the system at a rate λ according to a Poisson process.

Table 1: Summary of the notation used in Sections 4.1 $\sim$ 6.

Notation	Description
λ	event arrival rate (the number of events arriving per unit time)
λ_E	arrival rate of events of type E
μ	event service rate (the number of events served per unit time)
ρ	system utilization ($= \frac{\lambda}{\mu}$ [9])
QWT	queue waiting time (time spent by an event waiting in the input queue)
ST	service time (time taken to serve an event)
ST_E	service time of the event of type E
c_{ST}	coefficient of variance of ST ($= \mu\sigma_{ST}$, where σ_{ST} is the standard deviation of ST [9])
T_M	time taken to execute the merging
T_{PE}	time taken to evaluate one predicate (measured by experiments)
T_{RC}	time taken to construct (i.e., concatenate two partial pattern instances) one result (measured by experiments)
T_{BI}	time taken to insert one event into the buffer (measured by experiments)
Q	query event pattern
W	time window of Q
NP_{E_i,E_j} /$NP_{E_i;E_j}$	the number of parameterized/sequence predicates between event types E_i and E_j
SP_{E_i,E_j} /$SP_{E_i;E_j}$	total selectivity of parameterized/sequence predicates between event types E_i and E_j
$PI_{E_1;...;E_n}$	set of pattern instances matched to partial pattern $E_1;...;E_n$ from events arriving within W
$\lvert PI_{E_1;...;E_n}\rvert$	cardinality of $PI_{E_1;...;E_n}$ ($= \prod_{i=1}^{n}(\lambda_{E_i} \times W) \times \prod_{i=1}^{n-1}\prod_{j=i+1}^{n}(SP_{E_i,E_j} \times SP_{E_i;E_j})$)

- *G*: ST has a general distribution ($\mu = \frac{1}{E[ST]}$).
- *1*: A single server serves events one at a time according to a first-come, first-served discipline.

A *service* of an event is defined differently according to the event type. A service of an event of EEG is to eager-evaluate it, and that of LEG is to defer the evaluation. If the event is a final event, constructing pattern instances by merging is included in the service.

The output of the system is a set of pattern instances matched to Q. Such an output exists only when a *true final event*, which is defined in Definition 1, is served.

Definition 1. A final event is a *true final event* if there exists any pattern instance that contains the final event. Otherwise, the final event is called a *false final event*.

In this model, the expected latency $E[L]$ is as shown in Eq.(1). $E[L]$ is the average elapsed time from the point at which a final event arrives to the point at which all pattern instances that contain the final event are detected. $E[QWT_{TFE}]$ is the expected queue waiting time of a true final event. This time is caused by the events that arrive before the true final event arrives and the services for them. This time increases as ρ becomes higher and μ becomes slower [9]. $E[ST_{TFE}]$ is the expected service time of a true final event. Since this time includes the merging time, it increases as more event types are grouped into the LEG.

$$E[L] = E[QWT_{TFE}] + E[ST_{TFE}] \tag{1}$$

4.2 Latency estimation

We estimate $E[L]$ as shown in Eq.(2). We use the expected queue waiting time $E[QWT]$ instead of $E[QWT_{TFE}]$ in Eq.(1), and the expected service time of a final event $E[ST_{FE}]$ instead of $E[ST_{TFE}]$ in Eq.(1). While $E[QWT]$ and $E[ST_{FE}]$ are simpler to estimate, they have the same tendency as that of the $E[QWT_{TFE}]$ and $E[ST_{TFE}]$: $E[QWT]$ increases as ρ becomes higher and μ becomes slower [9], and $E[ST_{FE}]$ increases as more event types are grouped into the LEG.

$$E[L] = E[QWT] + E[ST_{FE}] \tag{2}$$

$E[QWT]$ can be calculated as shown in Eq.(3) [9]. We estimate μ, which is the inverse of $E[ST]$ as shown in Eq.(4). We estimate σ_{ST} which is the component of c_{ST} as shown in Eq.(5).

$$E[QWT] = \frac{\rho}{2\mu(1-\rho)}(1 + {c_{ST}}^2), \text{ where } \rho < 1 \tag{3}$$

$$E[ST] = \sum_{\forall E_i \text{ of } Q} \frac{\lambda_{E_i}}{\lambda} \times E[ST_{E_i}] \tag{4}$$

$$\sigma_{ST} = \sum_{\forall E_i \text{ of } Q} \sqrt{\frac{\lambda_{E_i}}{\lambda} \times (E[ST_{E_i}] - E[ST])^2} \tag{5}$$

$E[ST_{E_i}]$ in Eqs.(4) and (5) differs depending on which of *EEG* or *LEG* E_i belongs to. Suppose that the evaluation plan is $E_{E_1}E_{E_2}...E_{E_{|EEG|}}/E_{L_1}E_{L_2}...E_{L_{|LEG|}}$ where $|EEG|$ (or $|LEG|$) is the number of event types in EEG (or LEG). Then, $E[ST_{E_i}]$ is estimated as in Eqs.(6) $\sim$ (8).

$$E[ST_{E_{E_i}}] = \underbrace{(|PI_{E_{E_1};...;E_{E_{i-1}}}| \times \sum_{j=1}^{i-1} NP_{E_{E_i},E_{E_j}} \times T_{PE})}_{\text{the total number of predicate evaluations}} + \underbrace{(|PI_{E_{E_1};...;E_{E_{i-1}}}| \times \prod_{j=1}^{i-1} SP_{E_{E_i},E_{E_j}} \times T_{RC})}_{\text{the total number of results constructed}} \tag{6}$$

$$E[ST_{E_{L_i}}] = T_{BI} \tag{7}$$

$$E[ST_{FE}] = (E[ST_{E_{E_i}}] \text{ or } E[ST_{E_{L_i}}]) + T_M \tag{8}$$

If E is not the final event type, $E[ST_E]$ is estimated using either Eq.(6) or Eq.(7). If E is in the *EEG*, $E[ST_E]$ is estimated using Eq.(6). The first term estimates the time taken to evaluate parameterized predicates. The second term estimates the time taken to construct the result when all of the predicates are satisfied. If E is in the *LEG*, $E[ST_E]$ is equal to the time taken to insert the event into the buffer as shown in Eq.(7).

If E is the final event type, where E is in either the *EEG* or the *LEG*, $E[ST_E]$ is estimated using Eq.(8). The first term estimates the service time of the final event itself, which differs depending on the group to which the final event type belongs. The second term estimates the time taken by merging. T_M is estimated as the sum of the time taken to perform the join operations composing the merging. The time taken to perform each join operation is estimated using Eq.(9). The first term is the time taken to evaluate join conditions, and the second term is the time taken to construct the results of join operations when all of the join conditions are satisfied.

$$T_{E_i \bowtie E_j} = \underbrace{(|PI_{E_i}| \times |PI_{E_j}| \times (NP_{E_i,E_j} + NP_{E_i;E_j})}_{\text{the total number of predicate evaluations}} \times T_{PE}) + \underbrace{(|PI_{E_i}| \times |PI_{E_j}| \times SP_{E_i,E_j} \times SP_{E_i;E_j}}_{\text{the total number of results constructed}} \times T_{RC}) \tag{9}$$

As shown in Eq.(3), $E[QWT]$ is stable and can be calculated only if $\rho < 1$ [9]. If ρ of a candidate plan is greater than or equal to

one, we consider the $E[QWT]$ of that plan infinite, and thus, $E[L]$ of that plan is estimated to be infinite. If $E[L]$'s of all candidate plans are estimated to be infinite, we select the plan in which μ is the fastest. This is because the latency is heavily dependent on the throughput (i.e., μ) when $\rho \geq 1$, as noted in Section 1.

4.3 Heuristic intuition

Among the candidate plans, $E[ST_{FE}]$ decreases while $E[QWT]$ increases as the *eager evaluation ratio* (defined in Definition 2) of the candidate plan gets higher. $E[ST_{FE}]$ decreases because the cost of the merging decreases since more events are immediately evaluated even before the final event arrives. $E[QWT]$ increases because $E[ST]$ increases given that the matching order differs from the optimal order. On the other hand, $E[ST_{FE}]$ increases while $E[QWT]$ decreases as the *lazy evaluation ratio* (defined in Definition 2) of the candidate plan gets higher.

Definition 2. The *eager* (or *lazy*) *evaluation ratio* of an evaluation plan is the ratio of the number of event types in *EEG* (or *LEG*) to those in Q. An *eager* (or *lazy*) *evaluation plan* refers to the one whose eager (or lazy) evaluation ratio is 1.

Given the characteristics above, we state the heuristic intuition to select the optimal plan according to λ as follows, which will be used for sanity check of our proposed cost-based plan selection method in Section 6.

- Case 1: λ is slow ($\rho < \theta_{lower}$). It is best to use the eager evaluation plan. Because there is sufficient idle time, most events are evaluated using the idle time. In this case, the increase in $E[QWT]$ is shorter than the decrease in $E[ST_{FE}]$. Thus, the latency can be reduced by reducing $E[ST_{FE}]$.
- Case 2: λ is fast ($\rho > \theta_{upper}$). It is best to use the lazy evaluation plan. Because there is little idle time, most evaluations of events are delayed. In this case, the increase in $E[QWT]$ is longer than the decrease in $E[ST_{FE}]$. Thus, the latency can be reduced by reducing $E[QWT]$. If a prefix of the optimal matching order is same as the order specified in Q, it is better to group the event types of the prefix into the EEG because $E[ST_{FE}]$ decreases while $E[QWT]$ stays the same (ρ is the same) since the matching order does not change (i.e., stays the same as the optimal matching order).
- Case 3: λ is in the middle ($\theta_{lower} < \rho < \theta_{upper}$). The optimal evaluation ratio can be found by considering the trade-off between $E[QWT]$ and $E[ST_{FE}]$. An event type is grouped into the *EEG*, if grouped it reduces the latency since the increase in $E[QWT]$ is shorter than the decrease in $E[ST_{FE}]$. The lazy evaluation ratio of the optimal plan will become higher as λ becomes faster.

5. INTERMEDIATE RESULT MERGING

Whenever a final event arrives, the pattern instances matched to Q are constructed using partial pattern instances stored in the eager evaluation result buffer and events stored in deferred event buffers. Here, only those partial pattern instances and events within W are used to satisfy the time window condition of Q. We refer to this process as merging, which is formally defined in Definition 4.

Definition 3. Consider $E_1; E_2; ...; E_n$. For E_i and E_j (where $i < j$), a *sequence predicate* is $E_i.timestamp < E_j.timestamp$.

Definition 4. Suppose that the evaluation plan is $E_{E_1}E_{E_2}...E_{E_{|EEG|}}/E_{L_1}E_{L_2}...E_{L_{|LEG|}}$. *Merging* is $PI_{E_{E_1},...,E_{E_{|EEG|}}} \bowtie PI_{E_{L_1}} \bowtie ... \bowtie PI_{E_{L_{|LEG|}}}$ using the conjunction of the following predicates as a join condition:

- parameterized predicates grouped into *LEG*
- sequence predicates between an event type in *EEG* and the one in *LEG*
- sequence predicates among event types in *LEG*

Example 1. Figure 2(c) shows an example of merging detecting the query event pattern in Figure 2(a). The figure shows the state right after the event stream in Figure 2(b) arrived completely and the merging initiated by d_{15} is finished. In the join tree shown in the figure, leaf nodes represent the eager evaluation result buffer (green colored) and deferred event buffers (yellow colored), and internal nodes represent buffers into which the intermediate join results are materialized. First, the time window condition is evaluated prior to the start of the merging. (1) In this figure, a_1 and a_5, which are outside of the time window based on d_{15}, are discarded. When the merging starts, two-way join operations are performed repeatedly according to the predetermined join order. In this figure, the join order $((EERB_{BC} \bowtie DEB_D) \bowtie DEB_A)$ is used, in which the most selective join operation is performed first. Initially, (2) the partial pattern instances in $EERB_{BC}$ and DEB_D are joined, with the results being inserted into BCD. Next, (3) the results of prior join and pattern instances in DEB_A are joined, with the results being inserted into $ABCD$.

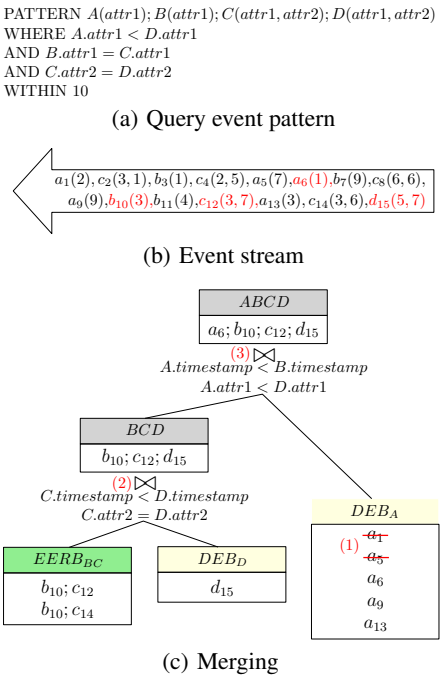


Figure 2: An example of the merging.

6. PERFORMANCE EVALUATION

In this section, we test the performance of **APAM**. We validate the proposed optimal plan selection method, and show that **APAM** achieves the lower bound of latency of the eager and lazy evaluation methods at any arrival rates.

6.1 Experimental setting

We conduct three experiments in which the optimal matching orders differ, as shown in Table 2. Next, we compare the performance of **APAM** with those of existing methods, i.e., the eager evaluation method and the lazy evaluation method. As the representative existing methods, we use SASE [1, 11, 13, 14] for the

eager evaluation method and ZStream [10] for the lazy evaluation method. SASE acts as **APAM** that uses an eager evaluation plan does, and ZStream as **APAM** that uses a lazy evaluation plan does.

Table 2: Three cases of experiments conducted.

No.	Description
1	The optimal matching order ($EDCBA$) is the reverse order of that specified in Q ($ABCDE$)
2	The optimal matching order ($ABCDE$) is identical to the order specified in Q ($ABCDE$)
3	Part of the optimal matching order ($BCDAE$) is identical to part of the order specified in Q ($ABCDE$)

The performance metric is latency. We measure the average elapsed time from the point at which each final event arrives to the point at which all pattern instances that contain the final event are detected. We conduct each experiment five times and use the average value.

For each experiment, we generate a synthetic event stream that consists of 100,000 events (50,000 for warm-up and 50,000 for measuring the performance). Events arrive into the system at arrival rate λ according to a Poisson process. For each experiment, we vary λ from 100 to 4500 events/sec. The event stream is a heterogeneous event stream that consists of five types of events. The distribution of event types follows a uniform distribution. The event schema is shown in Figure 3(a).

Figure 3(b) shows the query event pattern used in the experiments. The optimal matching order of each experiment in Table 2 is determined by selectivities of the parameterized predicates of this query event pattern. In the first experiment, the selectivities are 0.008, 0.006, 0.004, and 0.002. In the second experiment, they are 0.002, 0.004, 0.006, and 0.008, respectively. In the third experiment, the corresponding selectivities are 0.006, 0.002, 0.004, and 0.008. The selectivity of each parameterized predicate can be varied by varying the domain of the relevant event attribute.

eID (integer)
$type$ (integer, [0, 4])
$timestamp$ (long)
$attr1 \sim 9$ (integer)

(a) Event schema

PATTERN $A; B; C; D; E$
WHERE $A.attr1 < B.attr1$
AND $B.attr2 < C.attr2$
AND $C.attr3 < D.attr3$
AND $D.attr4 < E.attr4$
WITHIN 2000 events

(b) Query

Figure 3: The event schema and the query.

All experiments are conducted on a Linux machine equipped with an Intel Core i7 4790 CPU and 8 GB of main memory. We use SASE provided as an open-source software and implement ZStream as described in Mei, Y. et al. [10] These are implemented in Java.

6.2 Experimental results

6.2.1 Experiment 1

Figure 4 shows the results of the measurements of the latency of candidate plans for No. 1 in Table 2 while varying the event arrival rate. Figure 4 also shows QWT_{TFE} and ST_{TFE}, which are the time components of the latency. The x-axis represents the candidate plans, where the optimal plan selected by the proposed method is underlined.

As shown in Figure 4(a), when the event arrival rate is slow (where $\rho < 0.1$), the optimal plan selection method proposed in Section 4 selects the eager evaluation plan so as to utilize the idle time to the greatest extent possible. On the other hand, as shown in Figure 4(d), when the event arrival rate is fast (where $\rho > 1$), it selects the plan that uses the optimal matching order so as to increase the service rate. Here, the matching order of the plan selected is $DECBA$, which is another optimal matching order since selectivity of matching E after matching D is the same as that of matching D after matching E. As shown in Figures 4(b) and 4(c), when the event arrival rate is in the middle, the proposed method selects the plan that maximizes using the idle time provided that the queueing delay does not cancel the benefit. This result also conforms to the heuristic intuition in Section 4.3.

When the event arrival rate is 1500 event/sec, the proposed method selects the second optimal plan, but there is only a slight difference (1.30 times higher) between the latency of the selected plan and that of the optimal plan. In all cases, the latency of **APAM** which uses the plan selected is lower than or equal to the latency of the existing methods. When the event arrival rate is 100 events/sec, the latency of the proposed method is equal to the latency of SASE and is 27.94 times lower than that of ZStream. When the event arrival rate is 1500∼4500 event/sec, the latency of the proposed method is 15.73∼361.48 times lower than that of SASE and is 1.04∼1.17 times lower than that of ZStream.

6.2.2 Experiment 2

Figure 5 shows the results for No. 2 in Table 2. In this result, the proposed method selects the eager evaluation plan for all event arrival rate. Since the optimal matching order is identical to the order specified in Q, increasing eager evaluation ratio makes ST_{TFE} decrease while not affecting QWT_{TFE}. Thus, a plan with a higher eager evaluation ratio achieves a lower latency regardless of the event arrival rate. This result also conforms to the heuristic intuition. In all cases, the latency of **APAM** is lower than or equal to the latency of the existing methods. As the eager evaluation plan is selected for all event arrival rate, the latency of **APAM** is same with that of SASE, and 1.25∼5.71 times lower than that of ZStream.

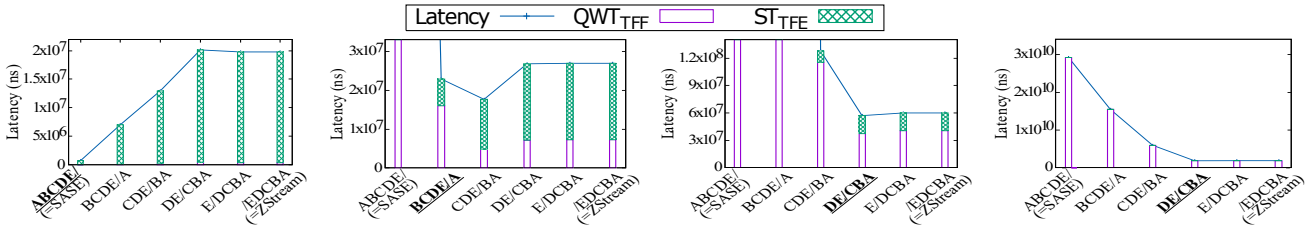
6.2.3 Experiment 3

Figure 6 shows the results for No. 3 in Table 2. As shown in Figure 6(a), when the event arrival rate is slow, the proposed method selects the eager evaluation plan. On the other hand, as shown in Figure 6(d), when the event arrival rate is fast, it selects the plan that uses the optimal matching order. As shown in Figures 6(b) and 6(c), when the event arrival rate is in the middle, the proposed method selects the plan that maximizes using the idle time provided that the queueing delay does not cancel the benefit. This result also conforms to the heuristic intuition.

When the event arrival rate is 4500 event/sec, the proposed method selects the third optimal plan, but there is only a slight difference (1.05 times higher) between the latency of the selected plan and that of the optimal plan. In all cases, the latency of **APAM** which uses the plan selected is lower than or equal to the latency of the existing methods. When the event arrival rate is 100 events/sec, the latency of the proposed method is equal to the latency of SASE and is 5.51 times lower than the latency of ZStream. For the rest of the case, the latency of the proposed method is 3.32∼60.09 times lower than that of SASE and is similar to or up to 1.61 times lower than the latency of ZStream.

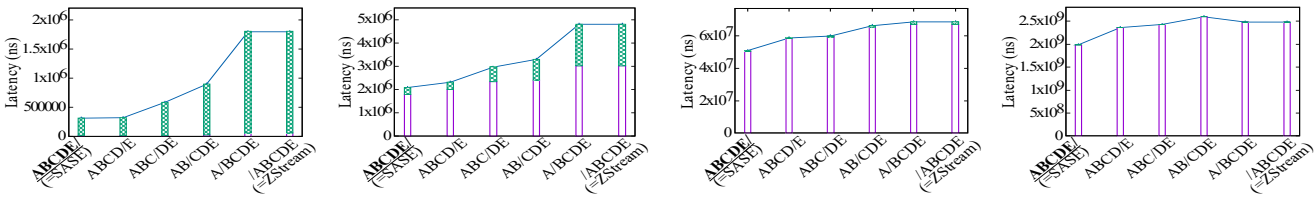
7. CONCLUSION

In this paper, we propose a hybrid eager-lazy evaluation method, called **APAM**, for low-latency event pattern detection. In **APAM**, event types and parameterized predicates of the query event pattern are partitioned into an eager evaluation group and a lazy evaluation group. When the final event arrives, pattern instances are constructed by merging partial pattern instances constructed from the eager evaluation group and events whose evaluations have been deferred in the lazy evaluation group. **APAM** can act as the eager evaluation method, the lazy evaluation method, or as a hybrid



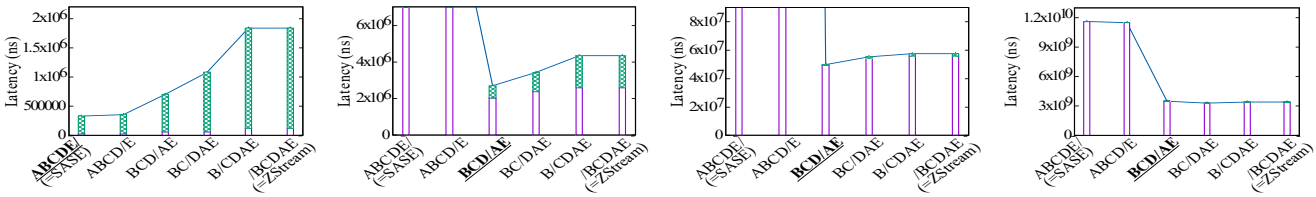
(a) 100 events/sec ($\rho < 0.1$) (b) 1500 events/sec (c) 3000 events/sec (d) 4500 events/sec ($\rho > 1$)

Figure 4: Latency of candidate plans with varying arrival rates in Experiment 1.



(a) 100 events/sec ($\rho < 0.1$) (b) 1500 events/sec (c) 3000 events/sec (d) 4500 events/sec ($\rho > 1$)

Figure 5: Latency of candidate plans with varying arrival rates in Experiment 2.



(a) 100 events/sec ($\rho < 0.1$) (b) 1500 events/sec (c) 3000 events/sec (d) 4500 events/sec ($\rho > 1$)

Figure 6: Latency of candidate plans with varying arrival rates in Experiment 3.

evaluation method according to the evaluation plan. By using the optimal evaluation plan, **APAM** can provide a lower-bound latency of those of existing methods at any event arrival rate. To find the optimal evaluation plan, we propose a cost model for estimating the latency of an evaluation plan. Based on the cost model, we propose a method that selects the optimal evaluation plan. Experimental results show that this method selects the optimal or near-optimal evaluation plan at varying arrival rates, which constitutes a lower-bound latency of existing methods. The results also show that **APAM** under the optimal evaluation plan can improve the latency by up to 361.48 times over the eager evaluation method and 27.94 times over the lazy evaluation method.

8. ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea(NRF) grant funded by Korean Government (MSIP) (No. 2016R1A2B4015929).

9. REFERENCES

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

[2] B. Cadonna, J. Gamper, and M. H. Böhlen. Efficient event pattern matching with match windows. In *Proc. ACM SIGKDD*, pages 471–479, 2012.

[3] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *Proc. VLDB*, pages 606–617, 1994.

[4] Q. Chen, Z. Li, and H. Liu. Optimizing complex event processing over rfid data streams. In *Proc. IEEE ICDE*, pages 1442–1444, 2008.

[5] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White. Towards expressive publish/subscribe systems. In *Proc. EDBT*, pages 627–644, 2006.

[6] R. Elmasri and S. Navathe. *Database Systems: Models, Languages, Design, and Application Programming*. Pearson, 2011.

[7] I. Kolchinsky, I. Sharfman, and A. Schuster. Lazy evaluation methods for detecting complex events. In *Proc. ACM DEBS*, pages 34–45, 2015.

[8] D. Luckham, W. R. Schulte, J. Adkins, P. Bizarro, H.-A. Jacobsen, A. Mavashev, B. M. Michelson, P. Niblett, and D. Tucker. *Event Processing Glossary – Version 2.0*. Event Processing Technical Society, 2011.

[9] J. Medhi. *Stochastic Models in Queueing Theory (2nd Edition)*. Academic Press, 2003.

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

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

[12] Y. Yan, J. Zhang, and M.-C. Shan. Scheduling for fast response multi-pattern matching over streaming events. In *Proc. IEEE ICDE*, pages 89–100, 2010.

[13] H. Zhang, Y. Diao, and N. Immerman. Recognizing patterns in streams with imprecise timestamps. *VLDB Endowment*, 3(1-2):244–255, 2010.

[14] H. Zhang, Y. Diao, and N. Immerman. On complexity and optimization of expensive queries in complex event processing. In *Proc. ACM SIGMOD*, pages 217–228, 2014.