META: Multi-resolution Framework for Event Summarization

Yexi Jiang[1], Chang-Shing Perng[2], Tao Li[1]
[1]Florida International University
[2]IBM T.J Watson Research Center

Abstract

Event summarization is an effective process that mines and organizes event patterns to represent the original events. It allows the analysts to quickly gain the general idea of the events. In recent years, several event summarization algorithms have been proposed, but they all focus on how to find out the optimal summarization results, and are designed for one-time analysis. As event summarization is a comprehensive analysis work, merely handling this problem with a single optimal algorithm is not enough.

In the absence of an integrated summarization solution, we propose an extensible framework – META – to enable analysts to easily and selectively extract and summarize events from different views with different resolutions. In this framework, we store the original events in a carefully-designed data structure that enables an efficient storage and multi-resolution analysis. On top of the data model, we define a summarization language that includes a set of atomic operators to manipulate the meta-data. Furthermore, we present 5 commonly used summarization tasks, and show that all these tasks can be easily expressed by the language. Experimental evaluation on both real and synthetic datasets demonstrates the efficiency and effectiveness of our framework.

1 Introduction

Event summarization is a process that mines and organizes event patterns to represent the original event sets, so that analysts can understand the system behaviors. Different from traditional frequent pattern mining techniques that simply discover patterns, event summarization provides a brief yet accurate summary for event datasets. These summaries smooth the learning curve of understanding the system and give the analysts insightful hints before conducting deep analysis with advanced data mining techniques.

Some research efforts have been working on providing various summarization methods [8, 13, 4, 9, 17]. Each of them defines its own way of summarizing events/documents. On the other hand, there are also some efforts [15, 3] working on providing various techniques for presenting event summarization results. From all these explorations, we can conclude that event summarization is not a problem that can be handled by a single model or algorithm. For different users or for different purposes, there are various ways of conducting event summarization, and also many parameters to be set. To obtain an event summary from different perspectives, an analyst has to re-preprocess the data and change the program time after time. This is a drain of analysts' productivity.

The predicament is very similar to that of the time when every data-intensive task has to use a separate program for data manipulation. The data representation and query problem were eventually addressed by the ER model and SQL. Following the historical path of DBMS and query languages, we believe event summarization (as well as event analysis) should also be abstracted to an independent software system with a uniform data model and an expressive query language.

An event summarization system has to be flexible enough, so that the real-life scenarios can be adequately and efficiently handled and supported. The followings are some typical scenarios that an event analyst would encounter.

SCENARIO 1. *An analyst obtains a system log of the whole year, but he only wants to view the summary of the events that are recorded between the latest 30 days. Moreover, he wants to see the summary without the trivial event "firewall scan". Also, he wants to see the summarization with the hourly granularity.*

SCENARIO 2. *After viewing summarization results, the analyst suspects that one particular time period of events behaves abnormally, so he wants to conduct anomaly detection just for that period to find out more details.*

SCENARIO 3. *The system has generated a new set of security log for the current week. The analyst wants to merge the new log into the repository and also to summarize the merged log with the daily granularity.*

To handle the work in the first scenario using existing event summarization methods, we need to perform the following tasks: (1) Write a program or use the existing program to extract the events occurred during the specified time range; (2) Write or leverage existing program to remove the irrelevant event types; (3) Write or leverage existing program to aggregate the events by hour; and (4) Feed the preprocessed events to existing event summarization methods to obtain the summary. Similarly, about the same amount of works are needed for the second and third scenarios. Note that, if parameter tuning is needed, a typical summarization

task requires hundreds of such iterations in the aforementioned scenarios. Therefore, it is inefficient and tedious.

Similar to OLAP as an exploration process for transactional data, event summarization is also a trial-and-error process for temporal event data. As event summarization requires repetitive exploration of the events from different views, we believe *it is necessary to have an integrated framework to enable users to easily, interactively, and selectively extract, summarize, and analyze the temporal event data.* Event summarization should be the first step of any other mining tasks, and its goal is to enable the analysts to quickly gain the general idea of the events. Similar to FIU-Miner [21] that supports rapid task configuration, the event summarization framework should allow analysts to easily compose various summarization and analysis tasks, and then to efficiently execute them.

To satisfy the above requirements, we propose an extensible event summarization framework called META to facilitate the multi-resolution summarization as well as its associated tasks. Instead of inventing new summarization algorithms, we focus on filling the missing component of the event summarization task and making it a complete knowledge discovery process. Therefore, our work is complementary and orthogonal to previous works that focus on proposing different event summarization algorithms.

We design META with the following principles: 1) the framework should be flexible enough to accommodate many real-life scenarios; and 2) the framework should ease the summarization tasks implementation as much as possible. Figure 1 shows the corresponding workflows of conducting the above scenarios with the META framework, including ad-hoc summarization, events storing, recovering, updating, and merging. For each scenario, the analyst only needs to write and execute a short piece of script.

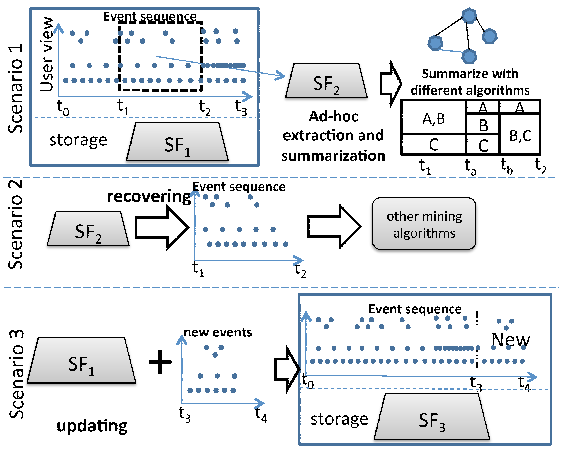


Figure 1: Summarization workflows of example scenarios

1.1 Contributions The contributions of this paper are listed as follows: (1) We present a multi-resolution data model called *summarization forest* to efficiently store the event sequences as well as the necessary meta-data. Summarization forest is designed to store and represent the event sequence in multi-resolution views with specified precision. (2) We define a summarization language which includes a set of basic operations for expressing summarization tasks. Each basic operation is an atomic operation that directly operates the data. (3) We introduce five commonly used event summarization tasks, including ad-hoc summarization, event storing, recovering, updating, and merging. We also show that these tasks can be expressed by the language. (4) We conduct a series of experiments on both real and synthetic event sequences to demonstrate the effectiveness, convenience, and efficiency of our proposed framework.

2 Related Works

Several works focusing on leveraging data mining and data processing techniques for event analysis have been proposed in recent years. According to the functionalities, they can be categorized as: *event log pre-processing*, *event summarization*, and *event based system analysis*.

In general, the event logs obtained from systems are unstructured/ semi-structured and are not immediately available for analysis. Researchers have proposed event format standards such as Common Event Expression [3] and Event Relationship Network [15] to describe all the event logs. Unfortunately, these representations are not widely adopted. In order to convert the raw logs into a canonical readable format, many efforts are needed to be made on pre-processing the logs [1, 11]. They utilize various techniques such as source code parsing, clustering and substring matching to extract the template from the raw event messages and then transform them into structured formats.

Event summarization focuses on extracting the high level overview from event log. Before directly diving into the details, it is a good choice for the analysts to see the summary first. Peng et al. [14] proposed an approach to find the dependency among events by measuring inter-arrival distribution of the event. Kiernan et al. [8] summarized the events by segmenting the event sequence according to the frequency changes. Wang et al. [13] further extended Kiernan's work by presenting the inter-segment relationship with HMM. Jiang et al. [4] provided a richer summarization of the events by providing the event relationship network (ERN) based on the logs, which captures the temporal dynamics. To the extent of our knowledge, existing research mainly focused on developing approaches to find the optimal summarization results, while our work is to present a comprehensive and extensible framework to facilitate the multi-resolution summarization for the system analysts.

Event log based system analysis focuses on revealing the hidden problems of the systems. Different analysis tasks pay attention to different application aspects, such as system failure tracing [19, 10], event correlation discovery [20, 18, 16], and event based trend analysis [5, 6, 7]. In practice, these methods are often conducted when the analysts already

have some prior knowledge about the data.

3 The Multi-Resolution Data Model

An event sequence can be represented in the form of event record sequence $D = (< t_1, e_1 >, < t_2, e_2 >, ..., < t_n, e_n >)$, where t_i is the time when an event occurs and e_i denotes the event instance with an associated 'type'. Each event instance belongs to one of the m types $\mathcal{E} = \{e_1, ..., e_m\}$. Note that the 'type' is a generic terminology. Any combination of the features of an event can be used as the 'type', e.g. the event category and event name in combination can be used as the event type. In this section, we first describe how to use an *event vector*, an intermediate data structure, to represent the event sequence. Then we introduce *summarization forest* (SF), the data model to store event sequences with multiple resolutions.

3.1 Vector Representation of Event Occurrences Given an event sequence D with m event types and time range $[t_s, t_e]$, we decompose D into m subsequences $D = (D_{e_1}, ..., D_{e_m})$, each contains the instances of one event type. Afterwards, we convert each D_i into an event vector V_i, where the indexes indicate the time and the values indicate the number of event occurrences. During conversion, we constrain the length of each vector to be $2^l, l \in \mathcal{Z}^+$, where l is the smallest value that satisfies $2^l \geq t_e - t_s$. In the vector, the first $t_e - t_s$ entries would record the actual occurrences of the event instances, and the remaining entries are filled with 0's. Example 1 provides a simple illustration on how we convert the event sequence.

EXAMPLE 1. *The left figure in Figure 2 gives an event sequence containing 3 event types within time range $[t_1, t_{12}]$. The right figure shows the conversion result of the given event sequence. Note that the original event sequence is decomposed into 3 subsequences. Each subsequence representing one event type is converted to a vector with length 16. The numbers in bold indicate the actual occurrences of the events, and the remaining numbers are filled with 0's.*

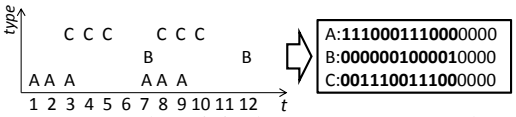


Figure 2: Convert the original event sequence to the vectors

Vectors intuitively describe the occurrences of events, but this kind of representation is neither storage efficient (as it requires $O(|\mathcal{E}|n)$) nor analysis efficient (as it does not support multi-resolution analysis). To facilitate the storage and analysis, we propose *summarization tree* to model the event occurrences of a single type. Furthermore, we propose *summarization forest* to model the event occurrences of the whole event log.

3.2 Summarization Tree The summarization tree is used to store the event occurrences for a single event type. It is capable of providing both frequency and locality of occurrences simultaneously. Moreover, it satisfies the multi-resolution analysis (MRA) [12] requirements by representing the event occurrences with various subspaces. This property enables the analysts to choose a proper subspace to view the data at a corresponding granularity. The summarization tree is formally defined below.

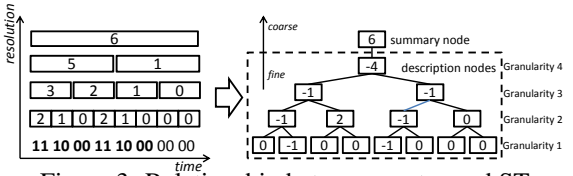


Figure 3: Relationship between vector and ST

DEFINITION 3.1. *A **summarization tree** (ST) is a balanced tree where all nodes store the temporal information about the occurrences of events. The tree has the following properties:*

1. *Each summarization tree has two types of nodes: summary node and description nodes.*
2. *The root is a summary node, and it has only one child. The root stores the total occurrences of the events throughout the event sequence.*
3. *All the other nodes are description nodes. They either have two children or no child. These nodes store the frequency difference between adjacent chunks (the frequency of the first chunk subtracted by that of its following chunk) of sequence described by lower level nodes.*
4. *The height of the summarization tree is the number of levels of the description tree. The height of a node in tree is the counted from bottom to top, starting from 0. The nodes at height i store the frequency differences that can be used to obtain the temporal information of granularity i.*

Considering event type A in Example 1, Figure 3 shows its vector and the corresponding summarization tree. As illustrated in the figure, the summarization tree stores the sum of the occurrences frequency (6 occurrences) at the root node, and the frequency differences (within the dashed box) in the description nodes at various granularities. Note that at the same level of the tree, the description nodes store the differences between adjacent sequence chunks at the same granularity. The larger the depth, the more detailed differences they store. For example, at granularity 1, every two adjacent time slots in the original event sequence are grouped into one chunk, and the grouped event sequence is '21021000'. Correspondingly, in the summarization tree, the frequency differences of each adjacent time slot $(0, -1, 0, 0, -1, 0, 0, 0)$ are recorded at the leaf level. Similarly, the frequency differences at various granularities are recorded in the description nodes at the corresponding levels.

It is clear that the space complexity of the summarization tree is $O(|T|)$, where $|T| = n$ and n is the length of the vector. From the storage perspective, directly storing the tree has no benefits for space saving. Basically, there are two ways to reduce the space complexity of summarization tree: *detail pruning* and *sparsity storage*.

3.2.1 Detail Pruning In practice, analysts only care about the high-level overview of the event occurrences. Consequently, there is no need to store all the details of the event sequences. As the summarization tree describes the event occurrences in a top-down manner — a coarse-to-fine strategy, we can save the storage by removing the lower levels of the description nodes. The pruned tree still contains enough details for analysis, and an analyst who analyzes a long-term event log would not care about the event occurrences at the second precision. Due to the hierarchical structure of the tree, we can reduce the storage space exponentially. Lemma 3.1 shows how much space can be reduced through pruning. For example, the original tree has a height of 14 levels and 8192 (or 2^{13}) nodes. If we prune the tree by removing the last 6 levels, the size of tree will become $\frac{1}{2^6}|T| = 128$, which is only about 1.5% of the original size. The pruned tree is still able to describe the event occurrences with 1-minute granularity.

LEMMA 3.1. *Suppose the height of summarization tree is H, if we only keep the nodes with a height larger than or equal to k, the size of the pruned tree is 2^{H-k}.*

Proof. According to the property 3 of the definition of summarization tree, besides the summarization node, the summarization tree is a perfect binary tree. If only the nodes with height larger than or equal to k are kept, the size of remaining nodes in perfect binary tree part is $\sum_{i=k}^{H-1} 2^{H-1-i} = 2^{H-k} - 1$. Therefore, the total size of the summarization tree after pruning is 2^{H-k}.

3.2.2 Sparsity Storage Another way to reduce the space is to only store the non-empty nodes of the tree. The majority of the event types rarely appear in the event sequence. In this case, the corresponding vector will be dominated with 0's. Accordingly, the transformed summarization tree will also contain many 0's. For example, event type X only occurs twice throughout a 2-hour (7200 second) event sequence. The first occurrence is the first second, and the second occurrence is the second second. The number of nodes in the corresponding summarization tree is 8192, but there are only 28 non-zero nodes. Lemma 3.2 provides a lower bound on how many zero nodes exist in a summarization tree.

LEMMA 3.2. *Suppose the occurrence proportion (the probability of occurrences at any time) of event type X is $r = \frac{\#X}{n}$, where n is the length of vector that stores the event occurrences. For the corresponding summarization tree, the proportion of zero nodes at height h is $p_h = \max(1 - 2^{h+1}r, 0)$.*

Proof. The proof can be found in Appendix.

Based on Lemma 3.1 and 3.2, we further show the space complexity of a summarization tree in Theorem 3.1.

THEOREM 3.1. *The space complexity of a summarization tree with granularity k is $O(\frac{|T|}{2^k} - \sum_{i=k}^{H} max(\frac{|T|}{2^h} - 2^{h+1}r, 0))$, where $|T|$ is the length of the vector, H is the height of the summarization tree, and r is the occurrence proportion as described in Lemma 3.2.*

Proof. The proof is based on Lemma 3.1 and Lemma 3.2. The number of nodes with the height (granularity) larger than or equal to k is $2^{H-k} = \frac{|T|}{2^k}$ according to Lemma 3.1. For each level $h \geq k$, the number of zero nodes is $n_h p = max(\frac{|T|}{2^h} - 2^{h+1}r, 0)$, and the sum of all nodes with height larger than or equal to k is $\sum_{i=k}^{H} \frac{|T|}{2^h} - 2^{h+1}r$. Therefore, the number of non-zero nodes in the summarization tree is $\frac{|T|}{2^k} - \sum_{i=k}^{H} max(\frac{|T|}{2^h} - 2^{h+1}r, 0)$.

It is true that the second term will become 0 when r is sufficiently large. However, based on the empirical study, most of the event types occur rarely, and therefore $0 < r \ll 1$.

3.3 Summarization Forest *Summarization forest* is a data model which contains all the summarization trees. In one forest, there are $|\mathcal{E}|$ summarization trees. Each stores the events of one event type. Besides trees, the summarization forest also stores the necessary meta-data. The summarization forest is formally defined in Definition 3.2.

DEFINITION 3.2. *A* summarization forest *(SF) is a 6-tuple $\mathcal{F} =< \mathcal{E}, \mathcal{T}, t_s, t_e, l, r >$, where:*

1. *$\mathcal{E}$ denotes the set of the event types in the event sequence.*
2. *$\mathcal{T}$ denotes the set of summarization trees.*
3. *t_s and t_e denote the start timestamp and end timestamp of the event sequence represented by $\mathcal{F}$.*
4. *l denotes the full size of each ST, including the zero and non-zero nodes. All the trees have the same full size.*
5. *r denotes the resolution of each ST. All the trees are in the same resolution.*

Note that since the summarization trees are stored in sparsity style, the actual number of nodes that are stored for each tree can be different and should be much less than the full size. Given a summarization forest, we can recover the original event sequences.

4 Basic Operations

In this section, we propose a set of basic operators which are built on top of the data model we proposed. These operators form the summarization language, which is the foundation of the event summarization tasks presented in our framework. The motivation of proposing a summarization language is to make the event summarization flexible and allow the advanced analysts to define the ad-hoc summarization tasks to meet the potential new needs.

The basic operators are categorized into two families: the *data transformation operators* and the *data query operators*. The operators of the first family focus on transforming data from one type to another, and they are not directly used for summarization work. The operators of the second family

Operation	Symbol	Description
Vectorize	$\circ(D_i)$	Vectorize the subsequence D_i.
Unvectorize	$\bullet(V_i)$	Unvectorize the vector V_i.
Encode	$\triangleleft(V_i)$	Encode V_i into a summarization tree T_i.
Decode	$\triangleright(T_i)$	Decode T_i back to vector V_i.
Prune	$\ominus(T_i)$	Prune the most detailed information of T_i.
Concatenate	$\mathcal{F}_1 \uplus \mathcal{F}_2$	Concatenate two SF $\mathcal{F}_1$ and $\mathcal{F}_2$.
Project	$\Pi_{e_{(1)},...}(\mathcal{F})$	Extract events of types $e_{(1)}, ..., e_{(k)}$ from $\mathcal{F}$.
Select	$\sigma_{[t_1,t_2]}(\mathcal{F})$	Pick the events occurs between time $[t_1, t_2]$.
Zoom	$\tau_i(\mathcal{F})$	Aggregate the events with granularity u.
Describe	Υ_{name}	Use algorithm *name* for event summarization.

Table 1: Notations of basic operations

focus on retrieving/manipulating data in read-only way, and they provide the flexibility of generating the summarization. To make the notations easy to follow, we list all the symbols of all these operations in Table 1. We will introduce their meanings later in this section.

4.1 Data Transformation Operators The data transformation operators includes *vectorize*, *unvectorize*, *encode*, *decode*, *prune*, and *concatenate*. Their functionalities are listed as follows:

Vectorize and **Unvectorize**: *Vectorize* is used to convert the single event type subsequence D_i into a vector V_i while *unvectorize* does the reverse work. Both of them are unary, and represented by symbol $\circ$ and $\bullet$, respectively. Semantically, these two operators are complementary operators, i.e. $D_i = \bullet(\circ(D_i))$ and $V_i = \circ(\bullet(V_i))$.

Encode and **Decode**: *Encode* is used to convert the vector V_i into a summarization tree T_i while *decode* does the reverse work. Similar to vectorize/unvectorize, *Encode* and *decode* are complementary operators and both of them are unary. We use symbol $\triangleleft$ and $\triangleright$ to denote them respectively.

Prune: The operator *Prune* is unary, and it conducts on the summarization tree. It is used to remove the most detailed information of the events by pruning the leaves of a summarization tree. Note that this operator is irrecoverable. Once it is used, the target summarization tree will permanently lose the removed level. We use $\ominus$ to denote this operator.

Concatenate: The operator *concatenate* is a binary operator. It combines two SFs into a big one and also updates the meta-data. We use $\uplus$ to denote this operation. Note that only the SFs with the same resolution can be concatenated.

4.2 Data Query Operators The data query operators include *select*, *project*, *zoom*, and *describe*. They all take a *summarization forest* $\mathcal{F}$ as the input. The data query operators are similar to the Data Manipulation Language (DML) in SQL, which provides query flexibility to users.
Their functionalities are listed as follows:

Project: The operator *project* is similar to the 'projection' in relational algebra. It is a unary operator written as $\Pi_{e_{(1)},e_{(2)},...,e_{(k)}}(\mathcal{F})$. The operation is defined as picking the summarization trees whose event types are in the subset of $\{e_{(1)}, ..., e_{(k)}\} \subseteq \mathcal{E}$.

Select: The operator *select* is similar to the 'selection' in relational algebra. It is a unary operator written as $\sigma_{[t_1,t_2]}(\mathcal{F})$.

Zoom: The operator *zoom* is used to control the resolution of the data. It is a unary operator written as $\tau_u(\mathcal{F})$, where u is the assigned resolution, the larger, the coarser.

Describe: The *describe* operator indicates which algorithm is used to summarize the events. Its implementation depends on the concrete algorithm and . all the previous event summarization papers can be regarded as proposing a concrete describe operator. For example, [4] summarize the events with periodic and inter-arrival relationships. The *describe* operation is written as $\Upsilon_{name}(\mathcal{F})$, where $name$ is the name of summarization algorithm used for describing the events. If necessary, the analyst can implement her/his own *describe* algorithm that follows the specification of our framework. In our implementation, the time complexity of all these operators are lower than $O(|\mathcal{E}||T| \log |T|) = O(|\mathcal{E}|n \log n)$.

5 Event Summarization Tasks

Considering the requirements of the analysts discussed in Introduction, we introduce five commonly used event summarization tasks: *summarization*, *storing*, *recovering*, *merging*, and *updating*, using the previously defined basic operators as the building blocks. The intention here is to demonstrate the expressive capability of the basic operators, instead of giving a thorough coverage of all the possible tasks.

5.1 Summarization Task Summarization task is the core of event summarization, and all prior works about event summarization focus on this problem. Based on the defined basic operators, analysts can summarize the events in a flexible way. In our framework, any summarization task can be described by the following expression:

$$\Upsilon_{name}(\sigma^*_{[t_1,t_2]}\tau^*_u\Pi^*_{E\in\mathcal{P}(\mathcal{E})}(\mathcal{F})).$$

The symbol $*$ denotes conducting the operation 0+ times. With the combination of operators, the analysts are able to summarize **any** subset of events in **any** resolution during **any** time range with **any** summarization algorithm.

One thing should be noted is that the order of the operators can be changed, but the summarization results of different orders are not guaranteed to be the same. For example, a commonly used implementation of the *describe* operator is based on the minimum description length principle [4, 8]. Such implementation aims to find a model that describes the events with least information. Therefore, the results of $\Upsilon_{name}(\tau_u(\mathcal{F}))$ and $\tau_u(\Upsilon_{name}(\mathcal{F}))$ are possibly different.

5.2 Storing Task Storing is an important task. Converting the raw event log time after time is time-consuming with low management efficiency. This task enables the analysts to convert the events into a uniform data mode only once and reuse it afterwards. The store task can be written as:

$$\mathcal{F} = \bigcup_{e_i\in\mathcal{E}_I} \ominus^*(\triangleleft(\circ(D_i))),$$

where $\mathcal{E}_I$ denotes the set of event types that the analysts are interested in, and $\bigcup$ denotes putting all the trees together to form the SF. The analysts are able to pick **any** time resolution and **any** subset of all the event types for storage.

5.3 Recovering Task Recovering task is the link between the event summarization and other data mining tasks. After finding the interesting piece of event logs via the summarization results, the analysts should be able to transform the selected portion of SF back to its original events, so they can use other data mining techniques for further analysis. The recover task can be expressed as:

$$\bullet(\triangleright(\sigma^*_{[t_1,t_2]}(\tau_u(\Pi^*_{E\in\mathcal{E}_I}(\mathcal{F}))))).$$

This expression shows that the analysts can selectively recover the piece of events with **any** subset of event types, at **any** time range and **any** time resolution.

5.4 Merging and Updating Tasks Both merging and updating tasks focus on the maintenance of stored SF, but their motivations are different.

The merging task is conducted when the analysts obtain the SFs with disjoint time periods and want to archive them altogether. Suppose $\mathcal{F}_1$ and $\mathcal{F}_2$ denote two SFs, where $\mathcal{F}_2$ contains more details (contains lower resolution level). The merging task can be expressed as:

$$\mathcal{F}_{new} = \mathcal{F}_1 \biguplus \ominus^*(\mathcal{F}_2).$$

As shown in the above expression, when we merge two summarization trees with different resolutions, the SF with higher granularity would be pruned to meet the SF with lower granularity. Then these two SFs would be merged with the *concatenate* operation.

Updating task is conducted when the analysts want to update the existing SF with a new piece of event log. It can be expressed by basic operators as follows:

$$\mathcal{F}_{new} = \mathcal{F} \biguplus (\bigcup_{e_i\in\mathcal{E}_I} \ominus^*(\triangleleft(\circ(D_i)))),$$

where the operand of $\bigcup$ is similar to the operand of $\bigcup$ in storing task. Firstly, the new set of subsequence D_i will be vectorized and then encoded into a SF $\mathcal{F}$. Then the new SF would be merged into the old SF same as the *merge* task.

6 Experimental Evaluation

We conduct a series of experiments to evaluate our proposed framework. In this section, we do not focus on demonstrating the meaningfulness or correctness of the summarization results, since it should be the work of the concrete summarization algorithm designers. Instead, the main goal of the evaluation is to explore the efficiency and the effectiveness of the proposed framework, and to show how META makes the summarization more flexible and convenient. More concretely, our experiments aim to answer the following questions: (1) What is the cost to store the events in the form of SF? (2) How efficient is it to retrieve and convert the data from the SF? (3) How effective and flexible can our framework support the event summarization? and (4) What about the performance of the updating and merging tasks?

In addition to the evaluation of META, we also give a case study to show how META facilitates analysts to conduct event summarization tasks. As a showcase, we leverage the algorithm proposed in [4] as the summarization algorithm, which summarizes the events from the perspective of inter-arrival temporal relationship.

6.1 Storage Cost To evaluate the storage cost of SF, we use several real event logs across different OS platforms and domains. These event logs are collected from customer's servers by IBM service department and the details of these logs are listed in Table 2 (Available at *http://share.olidu.com/events/*). These datasets are different in the aspect of time range, event occurrences, occurrences frequency, distinct event types, and log record styles.

Name	Domain	Time Units	#Types
secure-secure	Security	534,898	14
nokia-netview	Network	99,118,589	15
system-win	System	41,113,840	64
security-win	Security	5,579,292	35
application-win	Application	6,980,559	61

Table 2: Features of real datasets

Table 3 illustrates the occurrence proportion of the events in the real world datasets used in the experiments. We record the maximum, average, and minimum occurrence proportion of the event types in each dataset. Among all the datasets, the most frequent event type has the occurrence proportion 0.022, indicating the event occurs only 22 out of every 1000 time slots throughout the time range of the event sequences. The data in this table demonstrates that no event type occurs all the time (the occurrence proportion $r \ll 1$) in real world situation. Therefore, the second term of the O-notation in Theorem 3.1 is comparable to the first term, and it makes the theorem meaningful.

	Maximum	Average	Minimum
secure-linux	0.005	0.001	3.739×10^{-6}
nokia-netview	2.185×10^{-4}	4.064×10^{-5}	1.009×10^{-8}
system-win	4.886×10^{-4}	1.413×10^{-4}	2.432×10^{-8}
security-win	0.022	9.381×10^{-4}	1.792×10^{-7}
application-win	0.003	5.787×10^{-5}	1.432×10^{-7}

Table 3: Occurrence proportion in real datasets

In order to measure storage cost, we store the SFs as binary files using object serialization technology. We use the *compression ratio (CR)* to quantify the ratio of SF files comparing with the original log file, i.e., $CR = \frac{size(file)}{size(original\_file)}$. To further save the storage space, we leverage DEFLATE algorithm [2] to compress the serialized SF. Figure 4 shows the compression ratio of all the datasets. It can be observed that all the stored SFs cost less storage space comparing with the original logs ($CR < 1$). More-

over, after compressed by DEFLATE, even the worst compressed SF costs only 32.4% of the space of the original log file. This fact shows that storing the logs as SFs can save the disk space.

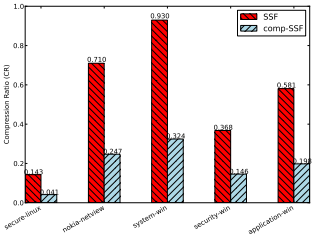


Figure 4: Compression ratio of SF and compressed SF

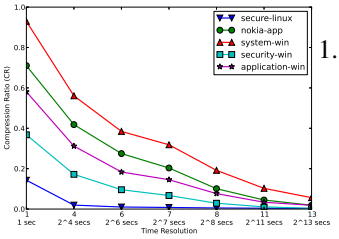


Figure 5: Compression ratio of SF in different resolution

The storage cost can be further reduced if the low level details are pruned. Figure 5 shows the compression ratio of each SF in different 7 resolutions without compression. Note that the values in the first row of x-label indicate the level of resolution we store the SFs, and the values in the second row indicate the corresponding approximate time resolution. As depicted in this figure, when the resolution is 13 (hourly resolution), even the most costly SF uses only 5% of space compared with the corresponding original file.

6.2 Efficiency Evaluation The efficiency evaluation is conducted in 3-fold. Firstly, we evaluate the performance of the summarization task by exploring different data query operators permutations. Moreover, we measure the performance of storing and recovering tasks to evaluate the time overhead of conducting event summarization within our framework. Finally, we investigate the performance of merging and updating tasks to evaluate the maintenance overhead.

We generate 15 synthetic datasets and investigate the performance of our framework on different datasets by changing 3 properties as listed in Table 4. The advantage of using synthetic datasets is that we can evaluate the performance of our framework with different properties systematically. Since here we only investigate the efficiency, the occurrences of events are randomly generated.

property	values	description
#types	20-100 step 20	The number of event types.
#events	60k-140k step 20k	The number of event occurrences.
#ts	10m-50m step 10m	The time slots in the time range.

Table 4: Properties of synthetic datasets

6.2.1 Performance of Summarization Task In this section, we evaluate the performance of all data query operators except *describe*. The reason is that the performance of *describe* depends on concrete summarization algorithms.

Similar to DML in SQL, the performance of the query varies with different operator permutations. To investigate how the order affects the query performance, we pick three sets of synthetic datasets to evaluate the time cost of different *project*, *select* and *zoom* permutations. In each set, we fix two properties and changes the third one. For *project*, we pick 10% of the event types from the SF. For *select*, we pick 10% of the time range, and zoom out the SF for one resolution. Table 5 shows the running time of all the 6 different permutations in 3 sets of experiments. By examining the experiment results in different perspectives, we can obtain following observations:

1. **Different operators have different time costs.** Table 5 shows that *select* is the most time-consuming and *project* is the most time-efficient. In our experiments, *select* is $10^2 \sim 10^4$ times slower than *zoom* and *project*. The reason is that by taking advantage of the SF, *zoom* only needs to remove all the leaves from trees in $O(\log |T|)$ time and *project* only needs to remove the useless trees in $O(|\mathcal{E}|)$ time. However, *select* is more complicated than the other two operators. It builds a new SF by extracting events satisfying the *select* parameters from the old SF, which takes $O(|T| \log |T|)$ time.
2. **Query performance varies drastically according to different operator orders.** The experiment results show that the fastest query costs only 3% the time of the slowest query on the same dataset. As mentioned before, *select* is the slowest operator. The more data it processes, the slower the execution would be. Therefore, the later the *select* operation is conducted, the shorter the query execution time would be.
3. **Query performance is insensitive to #events.** According to the experiment results conducted on the datasets with the same #types and #ts (1st group), the query performance appears to be stable when #events increases. On the contrary, the experiment results on the datasets with the same #events and #ts (2nd group) show that the query time varies linearly. Also, the results are similar for the datasets with the same #types and #events but with different #ts (3rd group).

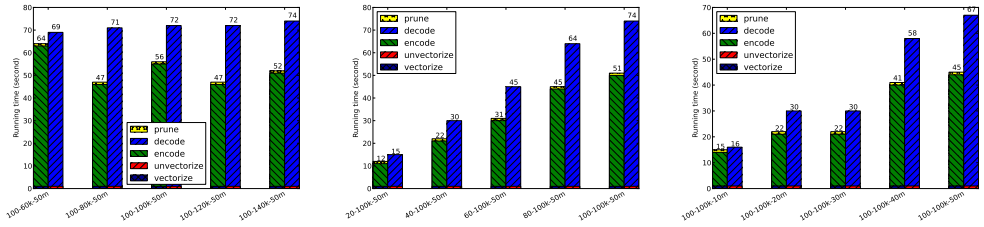
Based on the above observations, to avoid unnecessary time cost, a good query statement should postpone the select operation as much as possible. In our prototype, we conduct simple query optimization by reordering the operators.

6.2.2 Framework Time Overhead The tasks of storing and recovering are not directly related to event summarization, and they are considered as overhead for summarization. We conduct experiments on the same sets of datasets that are used in Section 6.2.1. For each datasets sets, we investigate the time cost of storing and recovering by revealing the running time of involved operators: *vectorize*, *encode*, *prune* for storing task and *decode*, *unvectorize* for recovering task.

Figure 6 shows the experiment results of the time overhead, where the first bar of each dataset indicates the overhead of storing task and the second bar indicates the overhead of recovering task. From the experiment results, we obtain two following observations. Firstly, all the experiments cost tens of seconds to finish the tasks. Due to the rare usage of these two tasks, the overhead is acceptable. Secondly, the time overhead of these two tasks are insensitive

Dataset \ Order	select-project-zoom			select-zoom-project			project-select-zoom			project-zoom-select			zoom-project-select			zoom-select-project		
	select	zoom	proj	select	zoom	proj	select	zoom	proj	select	zoom	proj	select	zoom	proj	select	zoom	proj
100-60k-50m	72.98	0.006	0.001	84.55	0.006	0.001	84.52	0.005	0.001	2.39	0.02	0.001	2.45	0.02	0.001	2.40	0.02	0.001
100-80k-50m	71.63	0.011	0.001	82.65	0.007	0.001	84.73	0.007	0.001	2.44	0.02	0.001	2.53	0.02	0.001	2.39	0.02	0.001
100-100k-50m	80.48	0.009	0.001	77.25	0.008	0.001	77.65	0.009	0.001	2.39	0.03	0.001	2.41	0.03	0.001	2.41	0.03	0.001
100-120k-50m	84.28	0.008	0.001	85.11	0.009	0.001	84.71	0.009	0.001	2.51	0.03	0.001	2.53	0.03	0.001	2.46	0.03	0.001
100-140k-50m	82.16	0.010	0.001	84.73	0.010	0.001	85.13	0.010	0.001	2.52	0.04	0.001	2.57	0.04	0.001	2.57	0.03	0.001
20-100k-50m	16.32	0.005	0.001	16.21	0.005	0.001	15.48	0.005	0.001	0.55	0.02	0.001	0.55	0.02	0.001	0.51	0.24	0.001
40-100k-50m	33.43	0.006	0.001	30.85	0.007	0.001	31.13	0.007	0.001	0.99	0.02	0.001	0.99	0.02	0.001	0.99	0.02	0.001
60-100k-50m	48.37	0.008	0.001	46.31	0.007	0.001	46.53	0.008	0.001	1.46	0.02	0.001	1.46	0.02	0.001	1.44	0.03	0.001
80-100k-50m	64.34	0.008	0.001	62.10	0.008	0.001	62.09	0.007	0.001	2.12	0.03	0.001	2.04	0.03	0.001	2.04	0.03	0.001
100-100k-50m	80.48	0.009	0.001	77.25	0.008	0.001	77.65	0.009	0.001	2.39	0.03	0.001	2.41	0.03	0.001	2.41	0.03	0.001
100-100k-10m	18.36	0.007	0.001	18.97	0.006	0.001	18.13	0.006	0.001	0.62	0.04	0.001	6.23	0.02	0.001	0.59	0.03	0.001
100-100k-20m	36.83	0.006	0.001	36.56	0.007	0.001	36.66	0.007	0.001	1.16	0.05	0.001	1.19	0.03	0.001	1.29	0.03	0.001
100-100k-30m	39.86	0.008	0.001	39.44	0.008	0.001	39.98	0.008	0.001	1.17	0.03	0.001	1.30	0.03	0.001	1.30	0.03	0.001
100-100k-40m	77.29	0.009	0.001	76.71	0.008	0.001	74.61	0.008	0.001	2.22	0.03	0.001	2.47	0.03	0.001	2.37	0.03	0.001
100-100k-50m	80.48	0.009	0.001	77.25	0.008	0.001	77.65	0.009	0.001	2.39	0.03	0.001	2.41	0.03	0.001	2.41	0.03	0.001

Table 5: Running time composition of different query orders (time unit: second)



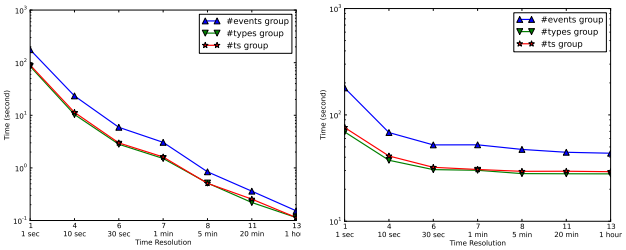
(a) Datasets with different #events (b) Datasets with different #types (c) Datasets with different #ts

Figure 6: Running time of storing and recovering tasks for datasets with different #events, #types, #ts

to #events but sensitive to #types and #ts. As we drill down to the operator level, we find that most of the increased running time comes from the *encode* operator in storing task and *decode* operator in recovering task. In our implementation, both of these two operations have the same time complexity $O(|\mathcal{E}||T|\log|T|)$. The running time would increase if either $|\mathcal{E}|$ or $|T|$ increases. Also, the distribution of event occurrences is another factor to affect the running time.

6.2.3 Performance of SF Maintenance In this section, we investigate the performance of maintenance tasks on two aspects: how the characteristics of events and how the resolution of data affects the performance. Similar to previous experiments, we evaluate the performance of both tasks using the same three groups of datasets. For each group, we convert the first dataset into a SF, and incrementally update and merge other datasets. Moreover, we evaluate updating or merging tasks by storing the SFs with 7 different resolutions. Figure 7(a) and 7(b) illustrate the results for merging and updating task. The time cost of the merging task is sensitive to the resolution but the updating task is not. In a high resolution, the merging task is more efficient than the updating task. The reason is that the updating task uses the time-consuming operator *encode* but the merging task does not.

6.3 An Illustrative Case Study To demonstrate how META facilitates summarization, we list 3 tasks (1st row) as well as the corresponding statement (2nd row) in Figure 8 to show how the analysts work on *security-win* dataset. We also attach corresponding summarization results (3rd row)



(a) Updating task (b) Merging task

Figure 7: Performance of maintenance tasks

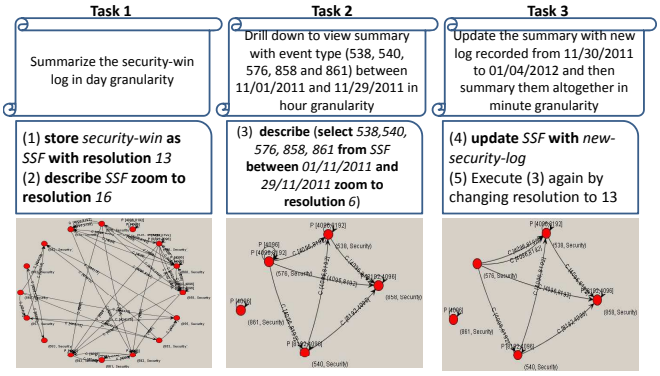


Figure 8: Summarize with META

by implementing the *describe* operator according to [4] [1].

As shown in Figure 8, the analysts only need to write one or two commands for each task. All the details are handled by the framework. Besides convenience, META also improves the reusability of data due to the SF's natural property. Once the security-win log is stored in SF, it is

[1] source code is available at http://users.cs.fiu.edu/ yjian004/#codes

directly available for all the 3 tasks, and there is no need to generate or maintain any intermediate data.

Without META, the analysts need to write programs on their own to conduct the data transformation and extraction. Taking task 2 for instance, the analysts should write several programs to transform the events in hourly resolution, to pick out the records related to the event types 538, 540, 576, 858, 861, and to extract the records occurring between 11/01/2011 and 1/29/2011. The analysts would do similar tedious work when facing the other two tasks.

7 Conclusions

We present our research efforts on establishing – META – an integrated event summarization framework. To facilitate the multi-resolution analysis of the events, we store the events in the form of *summarization forest*. Also, we propose a set of atomic operations on top of the data model and a set of summarization tasks to ease the work of the analysts. The experiment results demonstrated the efficiency and effectiveness of our proposed framework. For the future work, we will extend event summarization techniques to support distributed systems, which generate the events in more complicated environments at a much larger scale. Moreover, we will use the summarization results for automatic problem determination.

Appendix

Proof of Lemma 3.2:

Proof. We calculate the number of zero nodes from the bottom level to the top level. It is trivial to know that besides the root level, the number of nodes at height h is $n_h = \frac{|T|}{2^{h+1}}$. For each level, the number of zero nodes z_h equals to the number of nodes n_h minus the number of non-zero nodes u_h.

We start with $h = 0$ (the leaf level). In the worst case, the event occurrences are uniformly distributed along the time-line. There are two cases according to r:

1. $0 \leq r < \frac{1}{2}$. The event occurs in less than half of the time slots. In such condition, $u_0 = \min(r|T|, n_0)$, and $z_0 = n_0 - u_0 = \frac{|T|}{2} - r|T|$. So $p_0 = \frac{z_0}{n_0} = 1 - 2r$.
2. $\frac{1}{2} \leq r \leq 1$. The number of zero nodes at the leaf level can be 0. Since occurrences are uniformly distributed, it is possible that the event appears at least once in every two continuous time slots. In this case, $p_0 = 0$.

Therefore, the lower bound probability of the zero nodes at the leaf level is $p_0 = \max(1 - 2r, 0)$. When $h = 1$, in the worst case, the occurrences of non-zero nodes at the leaf level are still uniformly distributed, so $u_1 = \min(u_0, n_1)$. Therefore, $z_1 = n_1 - u_1 = \max(n_1 - u_0, 0)$ and $p_1 = \max(1 - 2^2 r, 0)$. When $h > 1$, if the occurrences of non-zero nodes at a lower level is still uniformly distributed, the number of zero nodes $u_h = \min(u_{h-1}, n_h)$. Similar to the case of $h = 1$, $z_h = n_h - u_h$, and $p_h = \max(1 - 2^{h+1} r, 0)$.

8 Acknowledgement

The work of Y. Jiang and T. Li is partially supported by the National Science Foundation under grants DBI-0850203, CCF-0939179, CNS-1126619 and IIS-1213026 and by the Army Research Office under grant number W911NF-1010366 and W911NF-12-1-0431.

References

[1] Michal Aharon, Gilad Barash, Ira Cohen, and Eli Mordechai. One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs. In *PKDD*, 2009.

[2] Salomon David. *Data Compression: The Complete Reference*. Springer, 2007.

[3] Common Event Expression. http://cee.mitre.org/.

[4] Yexi Jiang, Chang-Shing Perng, and Tao Li. Natural event summarization. In *CIKM*, 2011.

[5] Yexi Jiang, Chang-Shing Perng, Tao Li, and Rong Chang. Asap: A self-adaptive prediction system for instant cloud resource demand provisioning. In *ICDM*, 2011.

[6] Yexi Jiang, Chang-Shing Perng, Tao Li, and Rong Chang. Self-adaptive cloud capacity planning. In *SCC*, 2012.

[7] Yexi Jiang, Chang-Shing Perng, Tao Li, and Rong Chang. Cloud analytics for capacity planning and instant vm provisioning. *IEEE TNSM*, 2013.

[8] Jerry Kiernan and Evimaria Terzi. Constructing comprehensive summaries of large event sequences. *TKDD*, 3, 2009.

[9] Jingxuan Li, Lei Li, and Tao Li. Mssf: a multi-document summarization framework based on submodularity. In *SIGIR*, 2011.

[10] Zhenming Li, Shan Lu, Suvdar Myagmar, and Yuanyuan Zhou. Cp-miner: A tool for finding copy-paste and related bugs in operating system code. In *OSDI*, 2004.

[11] Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E.Milios. Clustering event logs using iterative partitioning. In *KDD*, 2009.

[12] Stephen Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE PAMI*, 11, 1989.

[13] Wang Peng, Haixun Wang, Majin Liu, and Wei Wang. An algorithmic approach to event summarization. In *SIGMOD*, 2010.

[14] Wei Peng, Chang-Shing Perng, Tao Li, and Haixun Wang. Event summarization for system management. In *KDD*, 2008.

[15] Chang-Shing Perng, David Thoenen, Genady Grabarnik, Sheng Ma, and Joseph Hellerstein. Data-driven validation, completion and construction of event relationship networks. In *KDD*, 2003.

[16] Ricardo Vilalta and Sheng Ma. Predicting rare events in temporal domains. In *ICDM*, 2003.

[17] Dingding Wang, Shenghuo Zhu, Tao Li, Yun Chi, and Yihong Gong. Integrating document clustering and multidocument summarization. *TKDD*, 5(3), 2011.

[18] Wei Xu, Peter Bodik, and David Patterson. A flexible architecture for statistical learning and data mining from system log streams. In *ICDM*, 2004.

[19] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Large scale system problem detection by mining console logs. *SOSP*, 2009.

[20] Kenji Yamanishi and Yuko Maruyama. Dynamic syslog mining for network failure monitoring. In *KDD*, 2005.

[21] Chunqiu Zeng, Yexi Jiang, Li Zheng, Jingxuan Li, Lei Li, Hongtai Li, Chao Shen, Wubai Zhou, Tao Li, Bing Duan, et al. Fiu-miner: a fast, integrated, and user-friendly system for data mining in distributed environment. In *KDD*, 2013.