ASEE 2014 Zone I Conference, April 3-5, 2014, University of Bridgeport, Bridgpeort, CT, USA.

Efficient Critical System Event Recognition and Prediction in Cloud Computing Systems

Yuanyao Liu
Department of Computer Science and Engineering
University of Bridgeport
221 University Avenue, Bridgeport, CT 06604, USA
yuaoyaol@bridgeport.edu

Zhengping Wu
Department of Computer Science and Engineering
University of Bridgeport
221 University Avenue, Bridgeport, CT 06604, USA
zhengpiw@bridgeport.edu

***Abstract*— A reliable system is one of most important goal of system engineers. However, system failure, software failure, outside attacks, and mis-actions make the system unstable and unreliable. Reducing the impact of system failure is possible if accurate failure predictions are provided. Resources, applications, and services can be scheduled around predicted failure and limit the impact. In Cloud Computing environment, virtualization is a basic technique that increases the utilization of system resources. However different virtual machines may generate number of system events. Events from different virtual machines can affect system stability together. Such mechanisms are especially important for Cloud Computing environment. In this paper, we propose an event recognition and prediction mechanism to increase system stability in a virtualization environment.**

Keywords—System Reliability, Event Recognition, Event Prediction, Virtualizations

I. INTRODUCTION

In the cloud computing environment, system reliability is an important factor that measures the wellness of the system. The system reliability in cloud computing environment does not only affect components in the system, it also affects other systems, which collaborate or cooperate with this system. System behaviors usually represent the system states and also system reliability. System behaviors indicate a set of series of system events happened in a time period. For example, the system crush event usually has some previous system events, such as memory overflow, hard failure errors, and even CPU temperature increasing. These events affect a system and prevent systems to provide certain services to users. Therefore, how to increase the system reliability is an important topic in cloud computing.

System behaviors consist of system event sequences. Some events happen all the time, every day, and even every minute. However, there are some events happen a few times, but can stop the system or make the system unstable. We consider these events as critical events. In order to increase the system reliability, one method is to prevent the critical events; another method is to heal the system after the critical event has happen.

Critical events usually appear in some frequent event patterns. These critical event patterns present system behaviors. System event patterns in distributed systems may not consist of the same machines, especially in virtualized data centers. Events from different virtual machines may works together to affect the reliability of systems, or may not affect the entire system. Virtual machines generate system events, which are collected by the system and recorded in the system logs.

Virtualization has been widely used in server clusters and data centers to build multiple services and increase utilization of hardware resources. Virtualization allows multiple operating systems running on a same machine. However, these operating systems need to be coordinated. The system resources are limited, so it cannot fulfill all need for virtualized machines at all the times. Although different resource requirements may increase the workload of the system, this is the goal of virtualization techniques.

Users cannot tell the difference between virtualized environments and bear hardware environments. Users always want to use the system efficiently. Therefore, virtualized systems receive all type of requests and commands. These requests and commands are finally passed to the host operating system or hypervisor to allocate system resources and do the computation. Host operating systems or the hypervisors are foundation of the upper level virtual machines. Events happened in virtual machines will affect host operating systems or hypervisors. These events may also affect each other in some ways. For example, the Netflix put its registration system on the Rackspace's virtual machines, on the same physical machine, there are also some small virtual machines running other customers' services. According to the Service Level Agreement, these virtual machines running on same physical machines can get enough resources that are stated in the agreement. However, the Netflix find out its registration system is not always stable. Other small virtual machines require some system resources that are not stated in the SLA explicitly. Lack of these system resources makes the Netflix's registration system virtual machine not stable. In this example, there must be some events happened in these virtual machines affects the reliability of the entire system.

Reliability is one of the most important aspects and requirement in cloud computing environment [13, 14]. Reliability is the proper functioning of the system under the

full range of conditions experienced in the field [1]. In order to increase the reliability of systems, there must be some mechanisms in systems can either avoid the system failures and faults or adjust systems to prevent the more serious failures and faults. Of cause, there is no system can ensure 100% reliability. System faults always happen in the entire computer systems. One lucky thing is we have log in most computer systems. When people design a computer system, people always want to keep some records about system events. These records are system logs. Nowadays, system logs are rich content databases. In papers [15] propose that the behavior of individual services in a service process must be monitored in order to settle any responsibility issue and meet the overall quality requirements from its customers. Similar in cloud computing systems, reliability is monitored by system monitors. System monitor has to try to avoid potential problems. For instance, in the RESERVOIR architecture [16], a service manager is responsible for monitoring the deployed services and adjusting their capacity in order to increase the reliability of the system. System monitor logs are used to learn such event sequences and event patterns. There is a lot of information in system logs. System logs also record critical events that cause a system fault or failure.

In order to predict critical events before they happen, we usually learn knowledge from historical system event data, and use this knowledge to predict system behaviors. In another word, we have to learning frequent system event sequences from the historical data. There are three steps in this process: learn frequent event patterns from history data, recognize frequent event patterns in run-time, and predict critical events in the run-time. Prediction of critical events is important for improvement of system reliability. For example there is a system event sequence data.

TABLE 1 EXAMPLE OF SYSTEM EVENT SEQUENCE DATABASE

Sequence Identifier	Sequence
1	CAABCE
2	ABCBD
3	CABCE
4	ABBCAE

In this system event sequence database example, there are 4 event sequences. Each sequence contains a critical event, for example, sequence 1 is ended with critical event “E”. There is another critical event “D” in sequence 2. System critical events are used to identify critical event sequences. There are some frequent event patterns in these event sequences. For example: “BCE” appears 3 times in this database. If there are “BC” event sequence appears, we can predict critical event “E” will probable happen in the future. In this paper, we propose a critical event pattern learning, recognition and prediction framework.

II. RELATED WORK

The sequential pattern mining problem was first proposed by Agrawal and Srikant in 1995 [2]. Sequential event pattern learning has been studied for decades [7, 8, 9, 11, 12]. Agrawal also presents an Apriori-based method in [3]. This algorithm is Generalized Sequential Pattern algorithm (GSP). GSP finds all the length-1 candidates (using one database scan) and orders them with respect to their support ignoring ones for which support < min_sup. Then for each level (i.e., sequences of length-k), the algorithm scans database to collect support count for each candidate sequence and generates candidate length-(k+1) sequences from length-k frequent sequences using Apriori. Some drawbacks of GSP are: a huge set of candidate sequences are generated, multiple scans of database are needed and it is inefficient for mining long sequential patterns as it needs to generate a large number of small candidates. In [4], authors developed a vertical format-based sequential pattern mining method called SPADE, which is an extension of vertical format-based frequent itemset mining methods. In vertical data format, the database becomes a set of tuples of the form <itemset: (sequence_ID, event_ID)>. The set of ID pairs for a given itemset forms the ID_list of the itemset. To discover the length-k sequence, SPADE joins the ID_lists of any two of its length-(k − 1) subsequences. The length of the resulting ID_list is equal to the support of the length-k sequence. The procedure stops when no frequent sequences can be found or no sequences can be formed by such joins. The use of vertical data format reduces scans of the sequence database. The ID_lists carry the information necessary to compute the support of candidates. However, the basic search methodology of SPADE and GSP is breadth-first search and Apriori pruning. Both algorithms have to generate large sets of candidates in order to grow longer sequences. In [5], authors use the concept of fuzzy sets to extend the original pattern discovery algorithms. These algorithms are used to discover fuzzy time-interval sequential patterns. In the web activities, the time interval is unpredictable, sometimes, the time interval is near the boundary that we predetermined. The idea of these algorithms is use the concept of fuzzy sets to overcome this boundary. There are Fuzzy Time Interval-Apriori algorithm and Fuzzy Time Interval-PrefixSpan algorithm. The basic idea of these two algorithms is inserts a pre-process to find all possible patterns. The Apriori and PrefixSpan algorithms are not modified. The space complexity is changed.

The paper [18] presents Temporal Pattern Search (TPS), an algorithm for searching for temporal patterns of events in historical personal histories. Comparing to the traditional method of searching for such patterns, which uses an automaton-based approach over a single array of events and sorted by time stamps, TPS operates on a set of arrays, where each array contains all events of the same type, sorted by time stamps. TPS searches for a particular item in the pattern using a binary search over the appropriate arrays. This algorithm used for search a pattern in an event stream. It needs predetermined patterns. This is an improvement of their previous work. In [19], authors propose a HAsh-based and PiPelIned (abbreviated as HAPPI) architecture for hardware enhanced association rule mining. They apply the pipeline methodology in the HAPPI architecture to compare itemsets with the database and collect useful information for reducing the number of candidate itemsets and items in the database simultaneously. In [20], authors use a level-wise association-

rule algorithm. It exploits anti-monotone and monotone constraints to reduce the problem dimensions level-wise. Each transaction, before participating to the support count, is reduced as much as possible, and only if it survives to this phase, it is used to count the support of candidate itemsets. Each transaction, which arrives to the counting phase at iteration k, is then reduced again as much as possible, and only if it survives to this second set of reductions, it is written to the transaction database for the next iteration.

Two principal approaches to critical event prediction based on the previous occurrence of failures can be determined: estimation of the probability distribution of a random variable for time to the next failure, and approaches that build on the co-occurrence of critical events [17]. In paper [21, 22, 6], authors build Markov models and N-gram to construct sequential classifiers. Markov models and Nth-order Markov models, when parameterized by a length of N, are essentially representing the same functional structure as N-grams. These algorithms use web logs to predict future events.

III. CRITICAL EVENT PATTERN LEARNING, RECOGNITION AND PREDICTION

In a cloud computing environment, multiple virtual machines usually run on one physical machine, or a service is divided into multiple virtual machines on multiple physical machines. Distributed service components may trigger number of events from various machines, including virtual machines and physical machines. Various types of system events are on behalf of system reliability and stability. For example, in Windows systems, there are 8 levels of events representing different importance of system events. The 0 level indicates the most critical events, which is usually system crush. The level 1 to level 7 events indicate from system overusing events, process errors to debug level events. The Table 1 shows a list of levels of system events. System event sequences usually start with normal events such as level 6 or level 5 events, and end with critical events, such as level 0 events or level 1 events. Table 2 show a list of windows event levels.

TABLE 2 SYSTEM AND SERVICE LOGGING LEVEL LIST IN WINDOWS

Level	System
0 Emergency: system is unusable	Hardware error
1 Alert: action must be taken immediately	System overusing
2 Critical: critical conditions	Process shutdown
3 Error: error conditions	Process abnormality
4 Warning: warning conditions	System access failure
5 Notice: normal but significant condition	Start of system process
6 Informational: informational messages	Hardware being online
7 Debug: debug-level messages	All system running related message

System event sequences construct system behaviors. Each system behavior represents series of system event patterns. These system events can be collected and put together to form a structured behavior. Like human behavior includes a series of actions, each system structured behavior has its own patterns. These patterns are good for us to analyze how a system performs. And we can predict system behaviors through learning history system event data.

Each structured behavior consists of number of features. These features are information of system states as well as system events. The most important feature is the time of an event happens. Event times can be used to form an order of different events. Different orders may represent different system behaviors. Therefore, event time is one feature that we have to keep in the data pre-processing. Event lasting time is another time-related feature in system event database. Some event may last certain time, because these events may take some time to finish it process.

Another important event feature is event level. As mentioned in previous, event levels indicate different event types. Different types of events appear in different position of event sequences. In another word, event levels also carry ordering information of event sequences. For example, Critical level events are usually treated as an end of event sequences. Event sources indicate which processes or components trigger these events. Event sources are usually used in event relationship analysis. A structured behavior represents a series of system events. Structured behaviors carry system information and event information.

A. System Event Pattern Learning

1) Summary of BIDE algorithm

According to previous presentation, system event sequences present behaviors of a system. Different structured behaviors have different event patterns. An event sequence ended with a critical event usually present a critical behavior. However, not every event sequence ended with critical events happens again and again. There are only a few event sequences appear frequently. We call these frequent event sequences frequent event patterns. Although the number of frequent event patterns is not very large, these frequent event patterns can have different prefixes of other events sequences. Therefore, if some event sequences happen and lead frequent event pattern appearing, we can recognize these frequent event pattern and predict critical event before it happens.

In order to predict critical system events, we first have to know under what kind of situation the critical system events happen. In another word, we have to learn what will lead to a critical system event. This situation can be presented as a frequent event pattern, which also presents a structured behavior of a system. Therefore, we can find out the knowledge of these structured behaviors or frequent event patterns from historical system event databases.

Learning a historical system event database is an ongoing research topic. There are number of techniques can find out frequent event patterns. In this paper, we customize an efficient mining algorithm. In [10], Wang and Han present an efficient algorithm for mining frequent closed sequences

without candidate maintenance. The proposed mechanism adopts a novel sequence closure checking scheme called BI-Directional Extension (BIDE) and prunes the search space more deeply by using the BackScan pruning method.

Time propriety of system events is an important feature of event sequences. Time represents an ordering propriety of events in a sequence. Therefore, events in an event sequence follow their orders, as well as in frequent event patterns. The example in the first section includes 4 event sequences, which includes 3 types of events. If we enumerate all event sequences, the sequence set will be: {A:4; AA:2; AB:4; ABB:2; ABC:4; AC:4; B:4;BB:2; BC:4; C:4; CA:3; CAB:2; CABC:2;CAC:2; CB:3; CBC:2; CC:2}. We can construct a pattern tree through this event sequence set. There are some short frequent pattern that can be treated as a prefix of its children, which means these node can become their children through add more events to it. For example, pattern "AB". Some of these frequent patterns have the same support as their children. Such as the node "AB:4" in the dataset, that has the same support with its children "ABC:4". Therefore, we can combine nodes "AB:4" and "ABC:4".

What kind of pattern can be absorbed by longer pattern? This is determined by Forward-checking and Backward-checking in BIDE algorithm. The Forward-checking is a method that eliminates event sequences S^p that its complete set of forward extension events is equivalent to the set of its locally frequent items whose supports are equal to $SUP^{SDB}(S^p)$. This is proved in the paper [10]. The Backward-check consists of a prefix event sequences, whether there is an existed item e^0 that appears in each of the i^{th} maximum periods of the prefix S^p in SDB, e^0 is a backward-extension event (or item) with respect to prefix S^p.

Any event pattern has Forward-extension or Backward-extension can be compressed into other frequent patterns, because it is a part of its Forward-extension or Backward-extension or both. The Forward-extension or Backward-extension in a tree is usually located in deeper levels. If there is neither Forward-extension event nor Backward-extension event of a frequent pattern, this frequent pattern can be called as frequent closed pattern. Frequent closed patterns indicate there is no further event patterns have the same supports. This feature reduces the search space of the pattern recognition and prediction.

Although the Forward-checking and Backward-checking generate a nonredundant frequent pattern space, this is not enough for reducing search spacing learning. Therefore, authors in [10] also introduce a "BackScan" Search Space Pruning method. This technique prunes some branches to further reduce the search space. Through this method, nodes frequent pattern tree cannot format a traditional tree, which means the learning process is speed up because there are no more branches. This tree structure has some draw backs in the recognition and prediction process, because of pruned nodes. There may not be a complete path from one node to its children. Therefore, during the prediction process, the path to a leaf node is needed that in order to predict critical event more early. The "BackScan" method prunes these nodes during the learning in order to reduce the search space, but we can keep such nodes during the recognition processes.

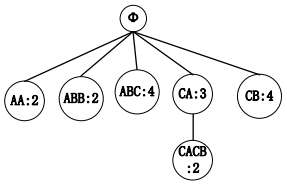

Figure 1 Frequent Closed Pattern Tree after Pruning

In the Figure 1, this tree is built use BIDE algorithm, there are only 6 nodes left in this tree. In this tree, all these nodes are frequent closed pattern, which can be used in further steps. It is a very simple tree which has only two levels. In the prediction, if the coming event sequence is "CA", we can directly predict the future events are "CB", because there is only one child under the "CA" node. If there is not any match an event sequence, for example "BC", in this tree, we cannot predict future events. Because there is no matching event patterns in this pruned tree. However, the event sequence "BC" is the suffix of frequent closed pattern "ABC".

2) Hash table with Frequent Pattern Tree

One most important feature or advantage of Hash table is the search speed, which is a constant time. A good hash function can greatly reduce search time in hash table search. In previous section, we summarize the BIDE algorithm which mines the event sequence database and find out all frequent closed patterns. It consists of Forward-checking, Backward-checking and BackScan pruning methods. This algorithm constructs a complete set of frequent closed pattern in an event sequence database, and a frequent closed pattern tree. The pattern tree is a compressed tree, where nodes not only contain their own information, but also contain their Forward-extension nodes and Backward-extension nodes. For example, the node "CABC" in Figure 1 is a leaf node, during the Forward-checking and Backward-Check, the event sequence "CAB" is compressed into the "CABC" node, because the node "CAB" has the same support as the "CABC" node. Therefore, one of important proprieties of this pattern tree is compressed information.

In order to map tree node to a hash table, we have to use hash function to calculate indexes for each node in the pattern tree. The hash function treats the event patterns as a string, because each event is represented as an English word. Therefore, we can utilize string hash function to map event patterns into a hash table. Features of each event, which are time, source, event level, and etc, are also used in the hash function. Through the time of events, we can form a time period for each frequent pattern. This time period usually formalize a time window for similar event sequences. And also the time period indicates system behaviors with reality world, because there may be some environmental factors affect the reliability of distributed systems. Sources and event level can also be used in the hash function that maps event pattern into hash table with event level orders. For example, if there are collisions in a hash table entry, the event pattern with higher event level can be put in the header of the event pattern

list. This could increase the speed of search for patterns that contain high event level events.

Each hash table entry links to a tree node in the pattern tree. In a hash table entry, there are also statistical data, and pattern time period. Statistical data is useful in pattern prediction. Pattern time period can be used to eliminate some unmatched event sequences or confirm recognition results. Hash table entry also contains an address of corresponding tree nodes. This address is the relationship between the hash table and frequent pattern tree. This information will be updated when the tree structure changed in run-time.

An advantage of using hash table is to fast locate nodes in the pattern tree. However, not every event sequence has a corresponding entry in the hash table. Event sequences, which do not have indexes in the hash table, are infrequent patterns or non-closed event patterns. These event sequences can be added to the frequent pattern tree. The hash table entry also will be updated accordingly.

IV. System Event pattern Recognition

System event pattern recognition is the second step of critical event prediction. In this step, we have to read an event sequences, and search frequent pattern database that try to find out a matched frequent pattern. If there is a matched frequent pattern, we also have to update the statistical data for further recognition and prediction.

In a hash table, to search an existed item, we can use hash function to directly calculate the index of this item. If the hash table entry with this index contains this item, it will report a match. If there is not a match in this step, we have to consider two situations. One situation is that the hash table entry of this index is empty, which means there is no such item can match given data item. Another situation is that there is a linked list in this hash table entry. If the given pattern can match any item in this linked list, we also can report a match. If there is no match in the linked list, we will report an unmatched notification.

If the given event sequence can be found in the hash table, which means this event sequence is a frequent closed pattern. And this item can be found in the event sequence tree. We will update the statistical data of this event pattern. The Figure 2 shows a process of recognition. In this example, the event sequence "ABC" has a hit in the frequent event pattern hash table. Then we follow the link to update the statistical data in the event sequence tree. This is a recognized sequence.

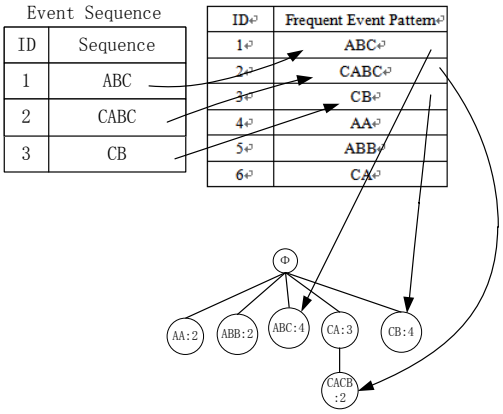

Figure 2 Event Sequence Recognition

If there is an unrecognized event sequence appears in the system, we have to update the event sequence tree, which include uncompress certain nodes to insert new event sequences. In order to uncompress a tree node in the event sequence tree, we have to create a new linked list, where the first node contains a "∅". This new linked list includes a path from "∅" to the target event sequences. For example, there is a new event sequence "ABCD", which does not have any match with all frequent patterns. We create a linked list with 5 elements {"∅", "A","AB","ABC","ABCD"}. Then we compare this linked list with event sequence tree, from the tail of the linked list. In order to compare elements in the linked list with frequent patterns in the event sequence tree, we use the hash function to calculate indexes. In this example, we can calculate that "ABC" is a frequent pattern in event sequence tree. Then, we create a child node of "ABC" and put the element after "ABC", which is "ABCD" to this child node. The event sequence tree become to the event sequence tree in the Figure 3.

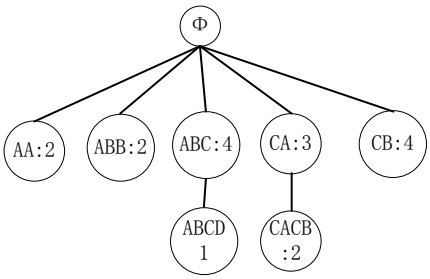

Figure 3 Updated Event Sequence Tree

In the traditional tree structure, parent nodes are connected with children nodes through links. However, in our approach, there is only one link from one parent node to its first child node. This child node is also connected with other children node through sibling links. The Figure 4 shows a structure of a node. Child node pointer points to its first child node; Sibling Node pointer points to its next sibling node; Parent node pointer points to its parent node; Next node pointer points to next node in the linked list. In following presentation, figures only show links that related to their topics.

Pattern	Child Node pintoer	Sibling Node Pointer	Parent Node Pointer	Next Node Pointer

Figure 4 Structure of a Node

In the new event sequence tree, the new node cannot become frequent pattern immediately, because the new event sequence does not meet the threshold yet. If there is an event sequence in the event sequence tree reach the support threshold, it will become a frequent pattern and be added in to the frequent pattern hash table. If the incoming event sequence is not in the frequent pattern but in the event sequence tree, we can use same process to update the event sequence tree, and check whether the incoming event sequence meet the support threshold. If the incoming event sequence does not match any frequent pattern in frequent pattern hash table and event sequences in the event sequence tree, the incoming event sequence will also be added into the event sequence tree. And update its statistical data.

The time is an important feature in the event pattern recognition process. Each event sequence carries its time information. This time information can be used in the recognition process. For example, an event sequence happens in time "t1", and end at time "t2", the time period is "Δt". In the recognition process, if there is not any frequent pattern match the incoming event sequence, the new event sequence should be add into the event sequence tree. After this event sequence, another event sequence comes into the system and carries the consecutive time information, which means these two event sequence happens in a tied time period. If the second event sequence is identical with the first event sequence, and these two event sequence come from a same source, these two event sequence may be a repeat event sequence. Therefore, we can identify whether there is a system jitter. The system should add this event sequence into the frequent pattern hash table for future recognition.

V. CRITICAL EVENT PREDICTION

The critical event prediction utilizes the statistical information of frequent patterns to predict known critical events. This step is processed at the same time of the recognition. Because every event pattern is ended with a critical event, the event sequence tree is a monitor of the system event sequences. At the beginning, the event sequence tree is a compressed tree, which does not contain all nodes, and its leaves are ended with critical events. Therefore, during the recognition, new added the event sequences are all ended with a critical event. As a result all leaves in the event sequence tree is ended with a critical event or a set of critical events.

The basic method to calculate probability of a critical event is to use Bayesian probability theorem. The Bayesian probability theorem provides a method to infer a probability of critical events. However, events in Bayesian theorem are independent to each other. The appearance of each event is base on its own statistical information. According to Bayesian theorem, the probability of appearance of an event is calculated by the (1).

$$P(y|x) = \frac{P(y|x)P(y)}{P(x)} \qquad (1)$$

In a cluster system or virtualized cloud computing system, most system events are not independent with other events. For example, one memory allocation request may cause a memory exhaust alert for other systems within the same physical machine. Therefore, there are tie relationships between events, especially for critical events. These relationships affect the result of prediction. Therefore, relationships have to be considered during the prediction. We use a constant "*C*" to indicate relationships.

$$P(y|x) = \frac{P(y|x)P(y)}{P(x)} + C \qquad (2)$$

The relationship constant "*C*" shows how interdependence is between two events. Constant "*C*" can be calculated through the structure of reverse pattern tree and relationship between critical event patterns (Formula 3). We assume the system is a reliable system, which critical events do not happen very frequently. Therefore, critical events appear only in the deeper level in the reverse pattern tree. Every prior event in a critical event pattern increases amount of relationship constant. The relationship constant includes a structured prediction function: "$f(x,y_i)$". The structure prediction function is a function Denote the output space for a given input pattern "x" as "$Y(x)$". We assume that we can define the output space for a structured example "x" using a set of constraint functions: "$f(x, y)$". This constraint function is very general. The output "y" is determined through the structure of pattern "x". The output of structure prediction function indicates the strength of relationship between child nodes of a pattern. For example, the structure probability of pattern "ABCD" is the relationship between its parent pattern "ABC" and its child nodes "ABCDA", "ABCDC" and "ABCDE". The result of this function is a number indicates the strength between "ABC and "ABCD". Relationship constant also includes an event relationship probability function "$g(x,y_i)$". The event relationship probability function indicates relationships between events. The temporal logic is used in this event relationship probability function. Relationships between events include direct and indirect relationships. These two types of relationships indicate different aspects of state transitions of systems. Direct relationships present the relationships of attributes of events, such as event "A" and event "B" are the same type of events. The event type attribute of "A", "B" is one of the direct relationships. The frequency of sequence "AB" appears in other patterns is another direct relationship of event "A" and "B". Indirect relationships represent relationships between events from one working node to the master node. Indirect relationships are used in the prediction in the master node. The master node has the entire event stream during the event processing. Although the master node controls functions that working nodes perform, master node cannot affect the exact events happened in working nodes. If an event from one working node result another event in another working node, these two events are indirectly connected. And this is the indirect relationship. Indirect relationships are learnt through pattern libraries. For example, event "A" and event "B" always appears together, and these

two events are from two different working nodes. Event sequence "AB" has an indirect relationship. Indirect relationships indicate likehood of events that have causal relationship.

$$C = f(x, y_i)g(x, y_i) \tag{3}$$

The structured prediction function works for both working nodes and master nodes. The relationship function includes two parts, direct relationship and indirect relationship. Direct relationship works for events in both working nodes and master nodes. Indirect relationship only works for master nodes, which include all events from upper level of the system since working nodes cannot know system events from other working nodes and master nodes.

VI. EXPERIMENT

The dataset we used in the experiment is records of cluster node outages, workload logs and error logs in Los Alamos National Laboratory. The data spans 22 high-performance computing systems that have been in production use at Los Alamos National Lab (LANL) between 1996 and November 2005. The data contains an entry for any failure leading to a node outage that occurred during the 9-year time period and that required the attention of a system administrator. For each failure, the data includes start time and end time, the system and node affected, as well as categorized root cause information. There are 23740 event records in this dataset. The first step is preprocessing. We clear the data with several features including event ID, event source, event start time, end time, and event type.

Through the BIDE algorithm, we can mining the frequent event pattern in this data set. The Figure 4 shows the result of BIDE learning result in different scale of events. In this test, the minimum support is set to 0.1%, the max length of frequent closed pattern is 10 events, which is also constrained by time. If time period between two events is over 24 hours, we will discard the early events. Source feature is also used to control the event sequence.

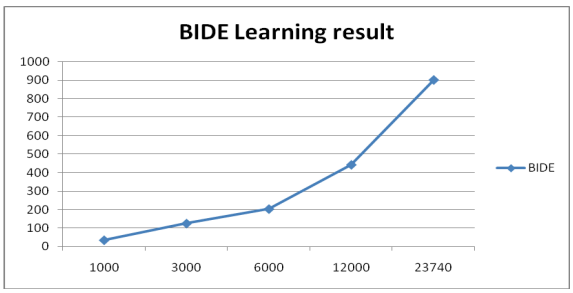

Figure 5 BIDE Learning Result

The learning process in this dataset shows in Figure 5. With the increasing of size of dataset, the time consumption in learning process is also increase, but it is increase in a liner fashion.

The Figure 6 shows the time that used for recognition in the same dataset. In this figure, the increase of processing time indicates there are still overhead in the program. The time include matched pattern and unmatched event sequences. The unmatched event sequence greatly increases the processing time.

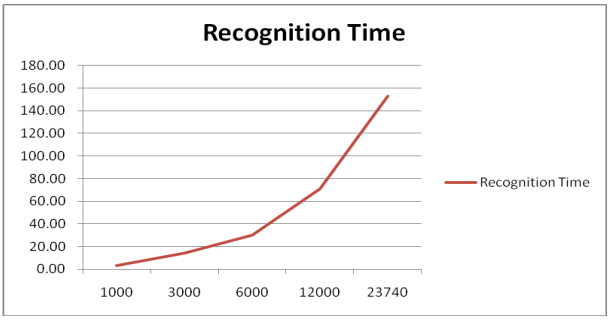

Figure 6 Hash Table Recognition Time

The Figure 7 shows the times that prediction process reported. With different thresholds, the report times are different. Higher thresholds reduce the search space of prediction process. Because the prediction process is triggered by the threshold, higher threshold setting triggers fewer prediction processes.

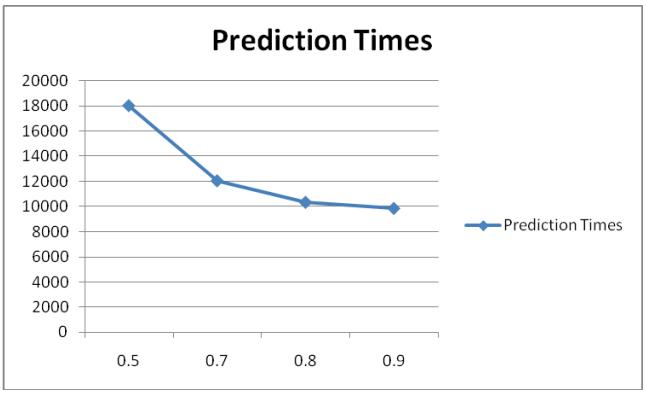

Figure 7 Hash Table Recognition Time

VII. CONCLUSION

In this paper, we proposed a critical event prediction framework in virtualization environment. The virtualization technique is widely used in distributed systems and very large data centers. In this framework, we use BIDE algorithm mining frequent closed pattern and store these event patterns in a hash table. Through the hash table, we reduce search time in the pattern recognition process. In the critical event prediction process, we use event sequence tree to calculate the probability of a critical event. This framework not only increases the frequent pattern recognition process, but also increases the accuracy of critical event prediction result.

References

[1] Clausing, Don, and Daniel D. Frey. "Improving system reliability by failure - mode avoidance including four concept design strategies." Systems engineering Vol 8, Issue 3, pp. 245-261 2005.

[2] R. Agrawal and R. Srikant, "Mining Sequential Patterns," Proc. Int'l Conf. Data Eng. (ICDE '95), pp. 3-14, Mar. 1995.

[3] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," Proc. Int'l Conf. Extending Database Technology (EDBT '96), pp. 3-17, Mar. 1996.

[4] M. Zaki, “SPADE: An Efficient Algorithm for Mining Frequent Sequences,” Machine Learning, vol. 42, pp. 31-60, 2001.

[5] Chen, Yen-Liang, and TC-K. Huang. "Discovering fuzzy time-interval sequential patterns in sequence databases." Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, Vol35, Issue 5, pp. 959-972, 2005

[6] J.Pitkow and P.Pirolli. Mining Longest Repeating Subsequences to Predict World Wide Web Surfing. In Second USENIX Symposium on Internet Technologies and Systems, Boulder, C0, 1999

[7] F. Masseglia, F. Cathala, and P. Poncelet, “The PSP Approach for Mining Sequential Patterns,” Proc. European Symp. Principle of Data Mining and Knowledge Discovery (PKDD ’98), pp. 176-184, Sept.1998.

[8] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.C. Hsu, “ FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining,” Proc. ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining (SIGKDD ’00), pp. 355-359, Aug. 2000.

[9] Derek Pao, Wei Lin, Bin Liu, "A memory-efficient pipelined implementation of the aho-corasick string-matching algorithm", ACM Transactions on Architecture and Code Optimization (TACO), pp 1-22, 2010,

[10] Wang, Jianyong, Jiawei Han, and Chun Li. "Frequent closed sequence mining without candidate maintenance." Knowledge and Data Engineering, IEEE Transactions on 19.8 (2007): 1042-1056.

[11] A.Paschke and A. Kozlenkov. Rule-based event processing and reaction rules. In Proceedings of RuleML, pages 53-66. Springer, 2009.

[12] Artikis, Alexander, Marek Sergot, and Georgios Paliouras. "Run-time composite event recognition." Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems. pp. 69-80, ACM, 2012.

[13] Wagle, Rohit, et al. "Distributed middleware reliability and fault tolerance support in system S." Proceedings of the 5th ACM international conference on Distributed event-based system. pp.335-346 , 2011.

[14] Atwa, Yasser M., and Ehab F. El-Saadany. "Reliability evaluation for distribution system with renewable distributed generation during islanded mode of operation." Power Systems, IEEE Transactions on, Vol 24, Issue 2 , pp.572-581, 2009.

[15] Yanlong Zhai, Jing Zhang, Kwei-Jay Lin, "SOA Middleware Support for Service Process Reconfiguration with End-to-End QoS Constraints", ICWS 2009, pp.815-822, July 2009,

[16] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Cáceres, M. Ben-Yehuda, W. Emmerich, F. Galán, “The reservoir model and architecture for open federated cloud computing”, IBM Journal of Research and Development archive Volume 53 , Issue 4, pp. 535-542, July 2009.

[17] Salfner, Felix, Maren Lenk, and Miroslaw Malek. "A survey of online failure prediction methods." ACM Computing Surveys (CSUR) Vol 42, Issue3, 2010.

[18] Wang, Taowei David, Amol Deshpande, and Ben Shneiderman. "A temporal pattern search algorithm for personal history event visualization." Knowledge and Data Engineering, IEEE Transactions on Vol 24, Issue 5, pp 799-812, 2012.

[19] Zhong, Ning, Yuefeng Li, and Sheng-Tang Wu. "Effective pattern discovery for text mining." Knowledge and Data Engineering, IEEE Transactions on, Vol 24, Issue 1, pp. 30-44,2012.

[20] Bonchi, Francesco, et al. "ExAMiner: Optimized level-wise frequent pattern mining with monotone constraints." ICDM 2003, Data Mining Third IEEE International Conference on. 2003.

[21] S. Schechter, M. Krishnan and M. D. Smith, Using path profiles to predict HTTP requests. In 7th International World Wide Web Conference, pages 457–467, Brisbane, Qld., Australia, April 1998.

[22] Su, Zhong, Qiang Yang, and Hong-Jiang Zhang. "A prediction system for multimedia pre-fetching in internet." Proceedings of the eighth ACM international conference on Multimedia, pp. 3-11, 2000