Volume 2, Issue 11, November-2015, pp. 785-790 ISSN (O): 2349-7084

International Journal of Computer Engineering In Research Trends

Available online at: www.ijcert.org

Study on PASS: A Parallel Activity-Search System

[1]MADIPADIGA VENKATESH, [2] V .RAMESH

[1]M.Tech (CS), Department of Computer Science & Engineering,

[2]Assistant Professor, Department of Computer Science & Engineering,

Sri Indu Institute of Engineering and Technology, R.R Dist Telengana, India.

Abstract: - In this paper we investigate on set of activities presented via temporal stochastic automata, partitions of activities based on level based events, in this connection our investigation on issues with activity creations on temporal multi-activity graph in order to address this issues as our proposed system how system used PASS architecture with various implementation parts with that coordinates computations across nodes in the cluster and also shown that this algorithms enables to handle both large numbers of observations per second as well as large merged graphs. And also shown Partitioning times vs. TMAG size for different partitioning schemes and TMAG densities (sparse-S, dense-D), averaged over number of compute nodes.

Keywords: Activity detection, temporal stochastic automata, parallel computation.

1. INTRODUCTION

In this paper, Given a set A of activities expressed via temporal stochastic automata, and a set O of observations (detections of low level events), Hidden Markov Models and Dynamic Bayesian Networks have been used extensively for representing activities [4]–[7]. A slight variant of these methods, stochastic automata, was used to represent activities in [3] and subsequently, a slight extension called Temporal Stochastic Automata was introduced [2], [8] showing that multiple stochastic automata can be merged together to recognize activities. We study the problem of identifying instances of activities from A in O. While past work has developed algorithms to solve this problem, in this paper, we develop methods to significantly scale these algorithms. Our PASS architecture consists of three parts: (i) leveraging past work to represent all activities in A via a single "merged" graph, (ii) partitioning the graph into a set of C sub graphs, where (C + 1) is the number of compute nodes in a cluster, and (iii) developing a parallel activity detection algorithm that uses a different compute node in the cluster to intensively process each sub graph. We propose three possible partitioning methods and a parallel activity-search detection (PASS Detect) algorithm that coordinates computations across nodes in the cluster. We report on experiments showing that our algorithms enable us to handle both large numbers of observations per second as well as large merged graphs. In particular, on a cluster with 9 compute nodes, PASS can reliably handle between 400K and 569K observations per second and merged graphs with as many as 50K vertices.

2. RELATED GROUND WORK

Hidden Markov Models (HMMs) and their variants have been used extensively in the past to model activities. Duong et al. [5] introduce the Switching Hidden Semi-Markov Model, a two-layered extension

of the Hidden Semi-Markov Model (HSMM). The bottom layer represents atomic events and their duration using HSMMs, while the top layer represents high-level activities in terms of atomic events. A survey of temporal concepts and data models used in unsupervised pattern mining from symbolic temporal data is presented in [14]. Automatic learning of transition probabilities in activity models is discussed in [15]. Finally, dynamic Bayesian networks [7] and Petri nets can also be used for tracking multi-agent activities. A probabilistic extension of Petri nets for activity detection is proposed in [16]. Context free grammars have also been used to define activities [17]. HMMs and their variants are the models more closely related to the temporal stochastic automata used in this paper, but our performance results significantly improve upon past work. Limitations of traditional database management systems in supporting streaming applications and event processing have prompted extensive research in Data Stream Management Systems (DSMS). An early yet comprehensive survey of relevant issues in data stream management was presented in [18]. Amongst the several systems resulting from research efforts in this direction, of particular relevance is TelegraphCQ [19], a streaming query processor that filters, categorizes, and aggregates flow records according to one or more CQL [20] continuous queries, generating periodic reports. Differently from traditional queries on static data collections, results of continuous queries on streaming data need to be periodically and incrementally updated as new data is received. A significant portion of research in this area has been devoted to optimization of continuous queries [21]. Other works target the recognition of events based on streams of possibly uncertain data [22]. Although the system we propose in this paper operates on streams of observation data, the scope of our work is drastically different from the scope of DSMSs. In fact, we are not interested in retrieving a set of data items satisfying (exactly) certain conditions and keeping this set up to date as new data items are received. Instead, we are interested in finding sets of records such that, with a probability above a given threshold, the records in each set together constitute the "evidence" that a given activity occurred in a specific time interval. Additionally, we want to track partially completed activity occurrences. To the best of our knowledge, DSMSs do not provide support for this type of probabilistic inference. Moreover, there has been limited work on efficient indexing to support probabilistic activity recognition. The aim of past work on indexing of activities was specialized to very specific activities as opposed to the very general temporal stochastic automaton based approach that we build on top of in this paper. For instance, Ben-Arie et al. [23] use multidimensional index structures to store body pose vectors in video frames. Kerkez [24] develops indexes for casebased plan recognition where knowledge about planning situations enables the recognizer to focus on a subset of the plan library containing relevant past plans. A two-level indexing scheme, along with incremental construction of the plan libraries, is proposed to reduce the retrieval efforts of the recognizer. In short, past work does not address the issue of indexing observations to find activity instances, and these indexing approaches do not account for uncertainty in what defines an activity. Most importantly, none of them is targeted to distributed scenarios when multiple compute machines are available over a network.

3. SYSTEM STUDY:

3.1. Presented System:

In this paper, we address the difficulty of scalable identifying instances of known activities (i.e., where activity models are known to the application developers such as in the cases listed above) in a high throughput stream of observations. We assume that activities are expressed as temporal stochastic (TS) automata, following the framework of [2] and its predecessor [3]. In particular, [2] took a set of known activity models expressed as temporal stochastic automata and merged them into a single graph and then proposed an algorithm to track activities in observation streams consisting of up to 28.5K observations per second on merged graphs consisting of fewer than 1000 vertices.

Real time applications::

1. Fraud in call data records
2. Online market place looks for fraudulent transaction in web transactions logs
3. Future situations of brokerage house

Pitfalls of Presented system:

1. It does not detect all fraud transactions efficiently.
2. Its provide the reliable solution

3.2. Proposed System:

In order to achieve this, in our PASS system we adopt a three-pronged approach illustrated in. We assume that

we start with an initially given set A of activities expressed as temporal stochastic automata.

Step1: In the very first step, shown in Fig. 1 with a 1 in a circle, we merge all of the activities in A into a single temporal multi-activity graph (TMAG). A TMAG captures all states and transitions present in any of the activities in A. TMAGs were first proposed in [2] which showed that merging graphs allowed multiple automata to be processed efficiently.

Step2: PASS implements activity detection on a compute cluster consisting of (C + 1) compute nodes or processors. What we try to do in the second step, shown in Fig. 1 with a 2 in a circle, is to partition the TMAG into C sub graphs. The idea is that one processor is used as a submit node and the remaining C processors are each assigned one of the sub graphs generated by partitioning. Splitting the TMAG allows us to scale the number of activities we can process as well as improves our processing time by using a compute cluster. Each component of the TMAG resulting from the split can be processed on an individual compute node in the cluster. We present three ways to partition a TMAG. The Minimal Overlap Partitioning (MOP) algorithm splits the TMAG by assigning to each vertex a "temporal extent". Intuitively, the temporal extent captures the period of time when the vertex can be active after the start of any activity in which that vertex is present. The intuition underlying minimal overlap partitioning is that if two vertices in a TMAG have similar temporal extents, then we should assign them to the same compute node. The Temporal Incidence Partitioning (TIP) method associates an "incidence" measure with any time interval. This incidence measure intuitively measures the number of vertices that can be active within a time interval. Some vertices, for instance, may occur at different time slices in different activities and as such, may have very wide time intervals. However, even with a very large temporal extent, the vertex may only be active infrequently within that temporal extent. TIP tries to split TMAGs by minimizing the standard deviation of these incidence measures. The Occurrence Probability Partitioning (OPP) algorithm transforms the TMAG into a weighted graph where the weights are learned by looking at actual observation (automaton state) streams to understand the true probability that one observation is seen after another observation. The idea is that if two observations occur consecutively very often within the real stream of observations being monitored, then these two observations often need to be processed very shortly after one another and hence, the corresponding two TMAG vertices should stay on the same compute node. OPP therefore weights edges in the TMAG using these co-occurrence probabilities and then partitions the TMAG using edge cuts after pruning away edges with very low weights.

Step 3: once the set A of activities we wish to detect are merged together into a TMAG and the resulting TMAG is partitioned across C different compute nodes, we are ready to process an observation stream and identify instances of the activities in A in the observation stream. To achieve this requires the core run-time component of the PASS system, shown in Step 3 (circle with a 3 embedded in it) in Fig. 1. The Activity-Search Engine automatically processes the observation stream using a decentralized algorithm that allows multiple nodes to concurrently process different portions of the observation stream with seamless handoffs occurring as needed from one compute node to another.

Advantages:

- Detect the frauds in different number of applications
- Reliable detection of all number frauds
- Quick detection is also possible here using the parallel activity search system

SYSTEM ARCHITECTURE:

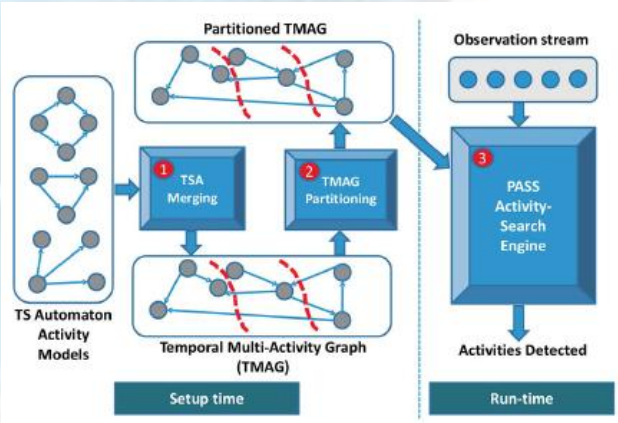

Fig 1. PASS architecture

4.1. PASS SYSTEM FUNCTIONING

- Design communication model
- Temporal stochastic automata
- Partitioning TMAG (temporal multiple activity graph)
- Parallel activity detection
- Performance evolution

Design communication model:

Our System implemented PASS in Java. As the implementation required balancing fast network communication and simplicity of use. When we were assessing the performance of the system by varying the number of compute nodes. First randomly creates a user specified number of vertices and then randomly generates outgoing edges based on a Gaussian distribution. Temporal activity graphs assume a temporal progression from a start node to an end node, that is, all paths through the graph have a temporal ordering.

Temporal stochastic autometa:

Hidden Markov Models and Dynamic Bayesian Networks have been used extensively for representing activities. A slight variant of these methods, stochastic automata, was used to represent activities in and subsequently, a slight extension called Temporal Stochastic Automata was introduced showing that multiple stochastic automata can be merged together to recognize activities. This section does not contain new material instead it recapitulates definitions first provided in.

A time span distribution specifies such transition probabilities, which may vary over time. In the following definition, a time interval is a closed interval of the set T of time points, which in turn can be assumed to be non-negative integers.

Partitioning TMAG (temporal multiple activity graph)

Two phenomena occur:

1) There may be thousands of known normal activities and as a consequence, TMAGs can be quite large, consisting of tens of thousands of vertices and hundreds of thousands of edges;

2) The number of observations made per second is very high, consisting of hundreds of thousands of observations per second. In this paper we propose techniques that exploit a cluster of (C+1) compute nodes by partitioning the set of vertices of a given TMAG G into C components so that each component can be separately processed by a different compute node. The additional compute node is used as a submit node. After building a partition P = {P1, . . . , PC} of G, node N(Pi) will thus handle all tuples f such that f .obs € Pi. We assume that each compute node includes an implementation of a sequential activity detection algorithm such as. Our framework is capable of working with any sequential activity detection algorithm within a node as long as the "inter-node" communications and handoffs are handled properly. In Section 4 we will discuss how this occurs in our system, where we employ our PASS Detect algorithm.

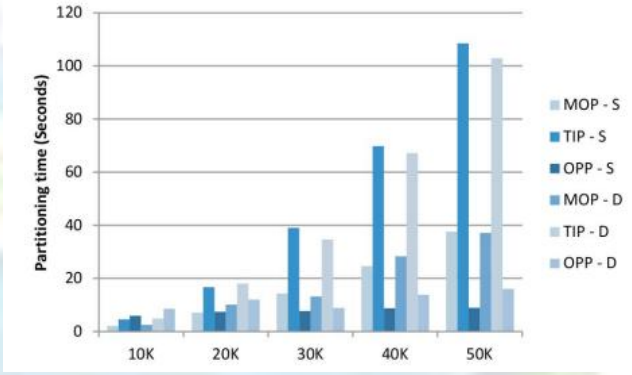

Fig 2. Partitioning times vs. TMAG size for different partitioning schemes and TMAG densities (sparse-S, dense-D), averaged over number of compute nodes.

Parallel activity detection:

When we have (C + 1) cluster compute nodes available in a cluster for activity detection, PASS uses one of those compute nodes as a submit node and the other C compute nodes each store the component Pi of a partition P = {P1, . . . , PC} of the TMAG G associated with a given set A of activities. Each compute node N(Pi) stores the restriction of G to the vertices in the component Pi, denoted G(Pi). Moreover, each compute node stores information about the set of frontier vertices w.r.t. Pi. A frontier vertex w.r.t. Pi is a vertex vj ∈ Pj with i _= j such that there exists a vertex vi ∈ Pi such that either (vi, vj) or (vj, vi) is an edge in the TMAG. When vj is a frontier node w.r.t. Pi, N(Pi) also stores the location of N(Pj). This way, during activity detection, if vj is observed, then a smooth handoff can be made to compute node N(Pj).

Algorithm PASS_Detect
Input: TMAG G_i, min activity probability p_t, boolean flag MP for the MP restriction
Global variable: List of activity instances F
Local variables: TMAG component P_i, association table T_i, observation sequence queue Q_i

```
1  if Q_i = ∅ then wait
2    for each a ∈ Q_i
3      t ← a.first
4      for each (t, id, t_p.t_0, prob, t_p) ∈ T_i
5        if a.prob · prob > p_t then
6          a' ← (t_p + a.trace, id, a.prob · prob)
7          if MP = true and ∃a_p ∈ Q_i ∪ F s.t. span(a_p) = span(a') and a_p.prob > a.prob
8          then discard a'
9          else
10           if(start_G(t.obs, id) ≠ ∅) then add a' to F
11           else
12             for each P_j ∈ t.obs.components s.t. o_p.obs ∈ P_j
13               Q_j ← Q_j ∪ a'
14             end for
15           end if; end if; end if;
16     end for; end for
```

Performance Evolution:

The performance of our partitioning schemes in terms of time to compute the partitions when varying TMAG sizes and number of compute nodes. OPP performs best in the majority of cases, with a performance gain that increases with larger TMAGs. On the other hand, it appears to suffer more than MOP and TIP4 from the density of large TMAGs. All partitioning schemes scaled well with the size of the input TMAGs, and their performance is almost independent of the number of partition size.

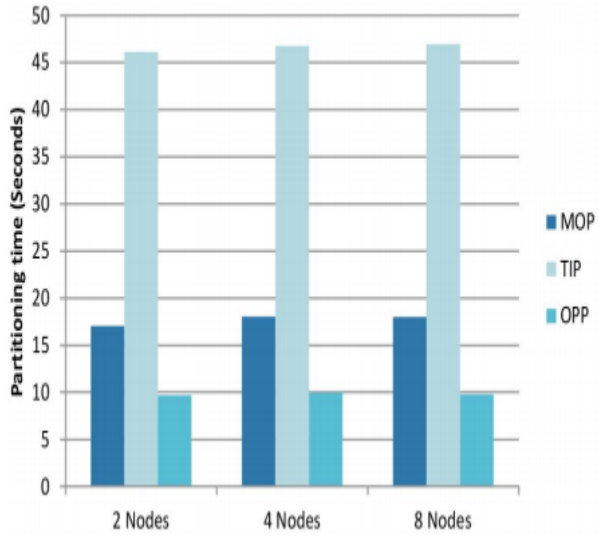

Fig 3.Partitioning times vs. numbers of compute nodes for different partitioning schemes, averaged over TMAG size and density.

5. CONCLUSION

In this paper we investigate on set of activities presented via temporal stochastic automata, partitions of activities based on level based events, and also review the PASS architecture with various implementation parts with that coordinates computations across nodes in the cluster and also shown that this algorithms enables to handle both large numbers of observations per second as well as large merged graphs.

REFERENCES:

[1] M. Albanese, V. Moscato, A. Picariello, V. S. Subrahmanian, and O. Udrea, "Detecting stochastically scheduled activities in video," in Proc. IJCAI, M. M. Veloso, Ed. San Francisco, CA, USA, 2007, pp. 1802–1807.

[2] S. Lühr, H. H. Bui, S. Venkatesh, and G. A. W. West, "Recognition of human activity through hierarchical stochastic learning," in Proc. PerCom., Fort Worth, TX, USA, Mar. 2003, pp. 416–422.

[3] T. Duong, H. Bui, D. Phung, and S. Venkatesh, "Activity recognition and abnormality detection with the switching hidden semi-Markov model," in Proc. IEEE CVPR, Washington, DC, USA, 2005.

[4] T. V. Duong, D. Q. Phung, H. H. Bui, and S. Venkatesh, "Efficient duration and hierarchical modeling for human activity recognition," Artif. Intell., vol. 173, no. 7–8, pp. 830–856, May 2009.

[5] R. Hamid, Y. Huang, and I. Essa, "ARGMode activity recognition using graphical models," in Proc. IEEE CVPR, Madison, WI, USA, 2003.

[6] M. Albanese, S. Jajodia, A. Pugliese, and V. S. Subrahmanian, "Scalable analysis of attack scenarios," in Proc. ESORICS, Leuven, Belgium, 2011, pp. 416–433.

[7] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," in Proc. FOCS, 1984, pp. 338–346.

[8] G. Palshikar and M. Apte, "Collusion set detection using graph clustering," Data Knowl. Eng., vol. 16, no. 1, pp. 135–164, 2008.

[9] M. Albanese, A. Pugliese, and V. S. Subrahmanian, "Fast activity detection: Indexing for temporal stochastic automaton-based activity models," IEEE Trans. Knowl. Data Eng., vol. 25, no. 2, pp. 360–373, Feb. 2013.