International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2016 IJSRCSEIT | Volume 1 | Issue 1 | ISSN : 2456-3307

Activity Search in the Big Volumetric Process of the Mining Automata using Parallel and distributed Computation

Sravan Kumar Vulchi[1] , Dr. Kalli Srinivasa Nageswara Prasad[2]
[1]M.Tech. CSE Student, GVVR Institute of Technology, Bhimavaram. Andhra Pradesh, India
[2]Professor, Department of CSE, GVVR Institute of Technology, Bhimavaram. Andhra Pradesh, India

ABSTRACT

Time and technology has its own role model with respect to the innovation. Technology and its model view to made things simpler for the end user; where the client need the pattern of the activity related to its domain. Information of extreme size diversity and complexity – is everywhere. This disruptive phenomenon is destined to help organizations drive innovation by gaining new and faster insight into their customers. Hence, in this paper we try to put the glimpse of the data search mechanism in order to use the stochastic automata to see the graph or in other from which may be relevant to the client. In this aspect we have used the parallel computing the logs which already mined and transaction data in various domains in order to give a statistical data to the end user. It can be used in both the way of prevention is better than care in order to make the things smarter and better way. In this paper we have considered both the automata theory to implement the stochastic automata using parallel computation giving raise the concept of efficiency, robustness and accuracy.

Keywords : Activity Detection, Temporal Stochastic Automata, Parallel Computation, Distributed Computing, Hadoop, Distributed File System

I. INTRODUCTION

Parallel computation promises shorter execution times or the ability to process greater quantities of data compared to sequential computation. However, in practice it is hard to realize a parallel implementation that comes close to achieving its theoretical potential. This is because efficient cooperation between processors is difficult to implement. Parallelism introduces a new set of concerns for the programmer: the scheduling of computations; placement of data; synchronization; and communication between processors. This adds greatly to the complexity to the programming task. An implementation must manage all these concerns in addition to computing a result. A skillful programmer can produce efficient implementations in such languages. However they are hard to use effectively; furthermore the code produced is often unclear, brittle and machine-specific.

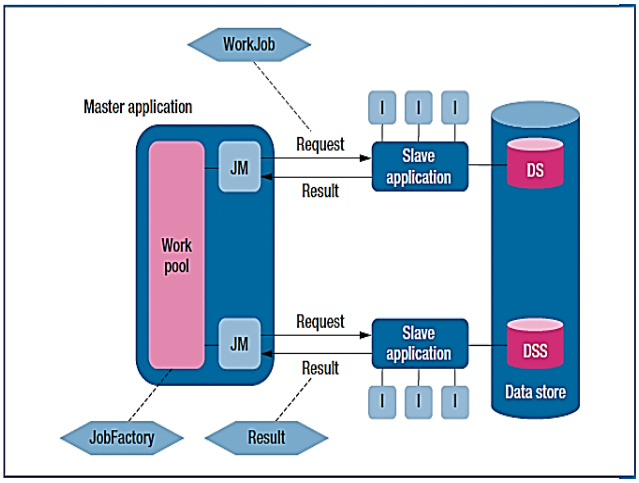

Figure 1.1. Illustration of the Parallel Computing

The weakness of these two approaches is that they present a single fixed level of abstraction. Implementing parallel algorithms is more complicated than implementing their sequential

counterparts, while at the same time the efficiency of the implementation is very important. This suggests a programming model that combines the benefits of both approaches: one that abstracts away from the complexity while still permitting fine control when necessary.

II. RELATED WORK

When performance of the support software is suboptimal the programmer will have difficulty in correcting the problem. Although a programmer may have the skill to produce a higher quality implementation, the abstractions of the parallel programming model may prevent them from doing so. It is sometimes possible to subvert abstractions when needed. However such work-around – dirty hacks result in the programmer fighting against the very feature that was intended to make the programming task simpler. Such work-around also diminishes the other benefits of programming with abstractions – such as transparency, safety, and portability.

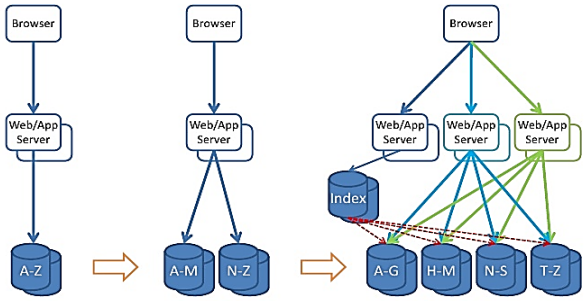

Figure 2.1. Model of the Sketch for the Data Process

The success of a parallel implementation can be assessed by a single measure: compare the runtime to that of an optimized sequential implementation. Because parallel processing is solely motivated by performance, it is often unacceptable to delegate implementation decisions to supporting software that may produce sub-optimal results. Instead the programmer may favor a programming model that provides the low-level control necessary to produce the best result. This is even though a programming model with few abstractions may make the programming task more difficult, the code harder to reason about and verify, and the resulting implementation harder to debug and maintain.

III. METHODOLOGY

Parallel programming languages and methodologies typically attempt to assist the programmer in one of two ways. The first approach is to provide layers of abstraction that hide the low-level details of the parallel machine from the user. This simplifies the programming task but reduces control over the finer details of the parallel implementation. Other languages provide as little abstraction as possible and require the parallelization concerns to be managed explicitly. Parallel machine architectures divide into two broad classes – shared memory systems and distributed memory systems. Shared memory machines are characterized by a set of processors that all have direct access to a common memory store, through which they may communicate. Distributed memory machines are comprised of a set of nodes interconnected by a network. Each node is a processor with its own local memory. Data is exchanged between nodes by exchanging messages across the network. Writing programs for distributed memory machines is considerably more difficult than implementing a similar shared memory program. Communication is via message passing, which introduces concurrency and possibly non-determinacy: in particular deadlock and race conditions are all possible. No determinacy greatly confuses reasoning about program behavior. The characteristics of the interconnection network – its latency and bandwidth – must also be considered. Failure to do so may cause processors that are waiting for a message to block excessively or the network to become saturated. The methodology is composed of a series of n stages, each of which has an associated language L_1; : : : ; L_n. Language L1 allows the expression of computations: all parallelization details are left unspecified. Each of the following languages in the series L_i ; i = 2; : : : ; n extends the previous language L_i 1 with constructs that make explicit the

implementation decisions of an additional parallelization concern.

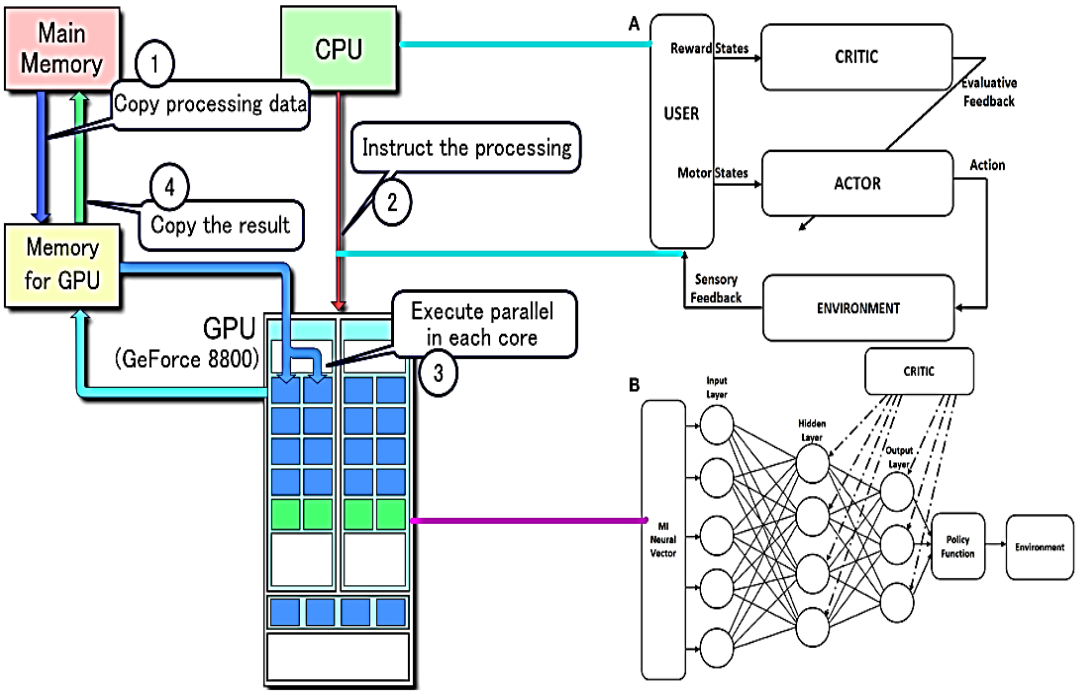

Figure 3.1. Architecture Design of stochastic automata of parallel computing

Therefore each language has a lower level of abstraction than its predecessor in the series. The process starts by expressing the computational portion of the algorithm as a program in language L1. Parallel implementation details are then incrementally introduced by rewriting this program in every language of the series in turn. Each transformation between stages only requires the programmer to make decisions about a single parallelization concern: the decision is supported by a language that presents an appropriate level of abstraction for that concern. The series of stages provides structure to the derivation. The introduction of parallel implementation details is ordered so that the higher-level, more fundamental decisions are taken before lesser concerns are tackled. By the time the program has been rewritten in language L_n all the parallelization details have been specified. A conventional implementation can then be produced with little further intervention from the programmer. We have designed and implemented a prototype of an incremental programming system.

3.1 Evaluation and Analysis

In all but the most embarrassingly parallel computations, some data will be computed on one processor and required by another. Communicating data across a distributed memory machine is expensive – the network has significant latency and limited bandwidth. Therefore parallel algorithms are designed to minimize the number of data redistributions required. The aim is to decompose the problem so that as much as possible of the data required by a processor is generated locally or on nearby processors. Another technique is to bundle together in the same communication different data that is to be redistributed in the same way. This may require adjusting the scheduling so that these results become available at the same time.

IV. CONCLUSION AND FUTURE WORK

Much research into language design and programming methodologies has been concerned with introducing models of computation that abstract away from the low-level machine details.

Whether it is an incremental change, such as the introduction of subroutines or the heap abstraction provided by C; or an innovation such as the execution model of Prolog; the aim is the same to simplify the programming task by hiding some of the complexity of the underlying machine. This is achieved by delegating the management of some of the implementation concerns to supporting software in the compiler or runtime system. As the programmer is released from the requirement to manage these concerns they are more able to concentrate on higher-level problem solving

V. REFERENCES

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

[2]. 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.

[3]. 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.

[4]. 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.

[5]. 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.

[6]. 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.

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

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

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

[10]. A. Guttman, "R-trees: A dynamic index structure for spatial searching," in Proc. SIGMOD Conf., B. Yormark, Ed. New York, NY, USA, 1984, pp. 47-57.

[11]. Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis, "R-trees: Theory and applications," in Advanced Information and Knowledge Processing. Secaucus, NJ, USA: Springer-Verlag, 2005.

[12]. N. Roussopoulos and D. Leifker, "Direct spatial search on pictorial databases using packed R-trees," in Proc. SIGMOD Conf., S. B. Navathe, Ed., New York, NY, USA, 1985, pp. 17-31.

[13]. D. R. Karger and C. Stein, "A new approach to the minimum cut problem," J. ACM, vol. 43, no. 4, pp. 601-640, 1996.

[14]. F. Morchen, "Unsupervised pattern mining from symbolic temporal data," SIGKDD Explor. Newslett., vol. 9, no. 1, pp. 41-55, Jun. 2007.

[15]. K. Seymore, A. McCallum, and R. Rosenfeld, "Learning hidden Markov model structure for information extraction," in Proc. AAAI Workshop Machine Learning for Information Extraction, 1999.

[16]. M. Albanese et al., "A constrained probabilistic petri net framework for human activity detection in video," IEEE Trans. Multimedia, vol. 10, no. 8, pp. 1429-1443, Dec. 2008.

[17]. V. Vu, F. Bremond, and M. Thonnat, "Automatic video interpretation: A novel algorithm for temporal scenario recognition," in Proc. IJCAI, San Francisco, CA, USA, Aug. 2003, pp. 1295-1302.

[18]. L. Golab and M. T. ozsu, "Issues in data stream management," SIGMOD Rec., vol. 32, pp. 5-14, Jun. 2003 [Online]. Available: http://doi.acm.org/10.1145/776985.776986