International Journal of New Technology and Research (IJNTR)
ISSN:2454-4116, Volume-4, Issue-11, November 2018 Pages 16-18

Performance Evaluation of Extended Latency Time Algorithm in different Linux based Operating Systems

Dipta Gomes, Aneem Al Ahsan Rupai, Mimun Barid, Abu Sufian

***Abstract* — Extended Latency Time (ELT) algorithm is an extension of the Latency time (LT) algorithm. Unlike LT, its extended version allows a system to assign tasks containing arbitrary time into the different processors. In doing so each task is assigned a time frame which decreases as each time unit passes. This report provides detailed information on the performance of ELT on different Linux based operating systems. The algorithm was implemented and the runtime was measured by providing graphs as input, in three different operation systems of Linux which are Ubuntu, Mint and Kali where average execution time in Kali Linux has been the highest which is close to 2.284 time units. From the three Ubuntu showed the most promising result which has shown an execution time of 2.198 time units. After some close observation it was found that the algorithm showed the best performance in Ubuntu.**

***Index Terms*— Extended Latency Time, Scheduling Algorithms, Algorithm Optimization**

I. INTRODUCTION

Shared memory is a way of making intercommunication between processes faster. Instead of communicating through the kernel the process share information in a shared space which is easily accessible by them [10]. As a result the computation time decreases. On the other hand, in a Distributed Shared Memory (DSM) there are multiple process with their own memory spaces. The processors are linked by an interconnecting network. If for any reasons one processor needs to access the other, requests (read/write) is made using the network which is similar to a data bus and responses are also given at the same way. DSM has the advantages of a decreased cost compared to other multiprocessor systems, provides a large virtual space, and it has better portability due to common programming interfaces [1]. Scheduling by means of which multiple processes are given access to memory for execution is very important. However it is very difficult to provide a proper allocation of the computer resources by scheduling [2].Over the past years there have been researches to solve this problem. Different types of scheduling algorithms have been proposed [9] [8]. One of them is ELT algorithm proposed in [3]. The main emphasis of the algorithm was to extend the latency time algorithm proposed in [4]. According to this algorithm any process will be broken down in smaller parts associating each with a time frame that can be updated. The synchronization time of the processes were also taken into account. However the ELT algorithm was not validated for optimization at different operating systems.

In designing a distributed system, a better choice for the intercommunication between the processes must be made. Shared memory can be thought of as a candidate for this matter as it provides a better understanding of the implementation procedure to the programmers. Michael Stumm and Sognian Zhon showed with implemented verifications that distributed shared memory has a competitive performance compared to data passing models which in some cases even out run the later one [5]. However scheduling different processes to run in such an environment is difficult. Martin, and Sanja proposed a framework that allows a decision maker to successively change the definitions of optimality criteria [6]. There have been huge amount of works done in order to address a scheduling algorithm that utilizes the computer resources properly. One such proposed algorithm is ELT and implemented by Irene Zuccar at. el and was verified to give a promising performance using DAG [3]. The algorithm was verified using heuristics, it was not implemented to demonstrate a practical performance.

There are different kinds of scheduling algorithm which operates in a similar way to ELT. One of them is 'List' that prioritize tasks, makes a list of the task and then assign them to available processors. Another one in Insertion Scheduling Heuristic that works like a list algorithm, but at first looks for empty time slots at the processor, and assigns tasks only if an empty slot is found [7]. The ELT algorithm on the other hand has the capability of assigning tasks whenever a processor is sitting idle regardless of whatever task is assigned to it [3]. As a result the computation speed increases.

This paper will provide information on the run time of the ELT algorithm on different operating systems. The algorithm will be implemented using C++ programming language. The reason for choosing C++ because it has a faster performance as the codes are typed checked before execution. In addition, it is a lower- level language, which enables the machine to convert the codes to machine language easily. Linux based

Dipta Gomes, Department of Computer Science, American International University-Bangladesh (AIUB), Dhaka, Bangladesh
Aneem Al Ahsan Rupai,, Department of Computer Science, American International University-Bangladesh (AIUB), Dhaka, Bangladesh
Abu Sufian,, Department of Computer Science, American International University-Bangladesh (AIUB), Dhaka, Bangladesh
Mimun Barid, Department of Computer Science, American International University-Bangladesh (AIUB), Dhaka, Bangladesh

operating systems have been chosen as the test environments for validation as these are open source and the algorithm can be easily incorporated to allow scheduling as the ELT describes. Operating systems like Windows or Mac does not allow this and hence testing the performance on those environments will be complex.

II. METHODOLOGY

ELT algorithm was implemented and the execution time in different operating system was compared. ELT is the extended version of LT algorithm and has been proposed for use in task with arbitrary time period. Figure-1 shows the steps to test the algorithm in details.

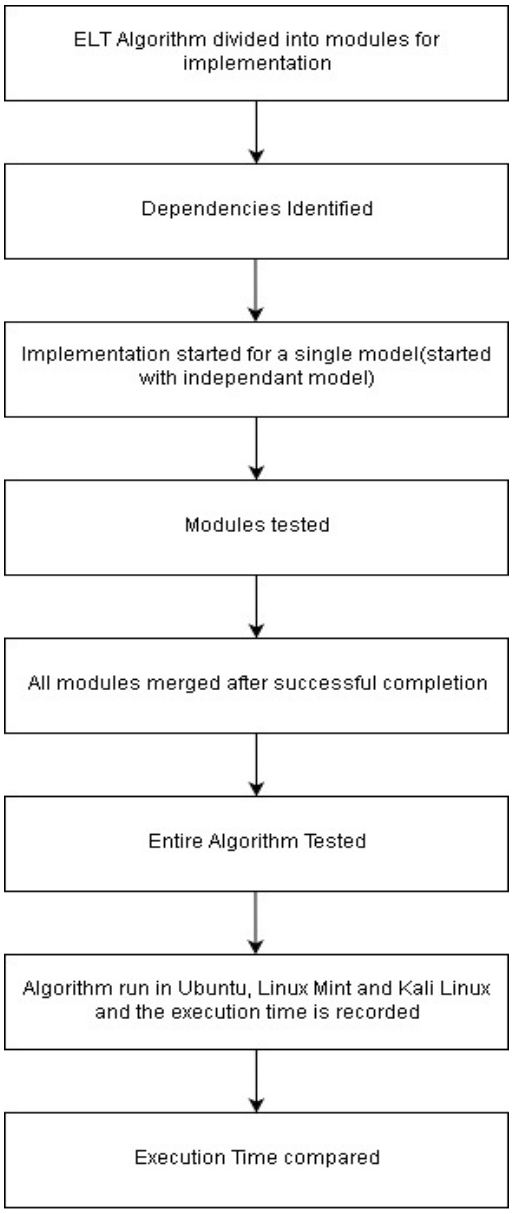

Figure 1: Flow chart for ELT

At first the algorithm was broken into several modules for ease of implementation. Then the requirements for each module were analyzed and the dependent and the independent modules were identified. The modules were set to be implemented first sequentially. After completion of each module, testing was done. All the modules were merged together after the implementations were completed. Then 6db6b6were used to test the performance of the entire algorithm. The performance was tested in two different operation environments which are Ubuntu and Kali Linux and a comparison between the execution time at different operating system was made

The implementation of the algorithm required the header files *limit.h*, *map*, *vector*, *list*, *ctime*, *queue* and *time*. For representation of DAG, a self-implemented algorithm is being used. Each vertex and edge was represented using a structure and a class for the graph. For representation of the vertices, each vertex was represented and traversed using map where the edge name was mapped with the vertex object. During prioritization, c++ stl function of priority queue and queue was used to track enabled nodes and scheduled task nodes.

The graph class contained method **addEdge** that added edges to the graph containing parameters source node name, destination node name and cost. The method **printPath** printed the path traversed containing only the destination node name. The function **dag** verified whether the function is Directed Acyclic graph or not. Here the function **getcostToEnd**for a specific vertex with name **startname** calculated the total cost from the vertices to the leaf nodes. **getcostToIn**calculates the total cost from the starting node to the specific node. A Function **get_Priority** calculated the priority of a specific node and the method **get_Piority_Of_All_Nodes** of all vertices. Finally the function ELT_Algorithm calculated and evaluated all the nodes based on their priorities.

Priority function: In the method, the parameter that it takes as input was Double system time, a variable that represented system time in time units. This was the input parameter in the algorithm that distinguished different classes of DAGs. A priority queue *pqueue* is maintained that kept the most prioritized task vertex at the top and sorted other vertices accordingly. Variables such as, ln, Ln, Li, Out, Hang and T was calculated from the graph vertices values. The final priority was then calculated using :

$Li = ln + Ln$

$P = \Sigma (Li + Out + Hang + T)$

In the ELT algorithm function first the priority of the nodes was calculated and the time ‘t’ was taken as double t_units=0 . Then the time window of vertex was assigned equal to the size of the task. All the nodes in the graph was then inserted to a vector Unsched for indicating unscheduled. All the nodes with t_level =1* was then enabled and insered into another vector Enabled. After each time window the unit of the task is processed and each task was ended when its time_window<0 and was enabled. Hence all the tasks were ended within an extended latency as long as their time_window is <0.

III. RESULTS AND DISCUSSION

The ELT algorithm was implemented across three different Linux platforms where the Prioritization function was implemented using the basis of the summation of longest path, critical path, longest path through the nodes, number of tasks achievable by the nodes and duration of each node. Here the Critical Path distance was added with the subscript of the subsets (I, OUT, HANGs, Ts) to which each task belonged.

Through implementation of the algorithm across various devices the total execution time of the algorithm was recorded. The set of tests used consisted of 100 DAGs which were divided into five categories: 20 of which had tasks with random duration (CR), 20 had tasks with duration between 1 and 2 time units (CR1-2), and 20 had tasks with duration between 1 and 5 time units (CR1-5); and 20 DAGs of "known structure" 23 had tasks with arbitrary duration (CC) and 23 had the same structure of the previous ones, but the tasks had durations between 1 and 2 time units (CC1-2). The average of the execution time of the algorithm is calculated and recorded. It was found that Kali Linux took the longest time for execution of the algorithm and Ubuntu the least time.

	CR	CR(1-2)	CR(1-5)	CC	CC(1-2)
Linux Mint	2	1.22	3.2	2.8	2.2
Kali Linux	1.91	1.21	3.14	2.6	2.3
Ubuntu	1.96	1.21	3.14	2.55	2.13

Table 1 – Runtime results of ELT in different Linux based operating system

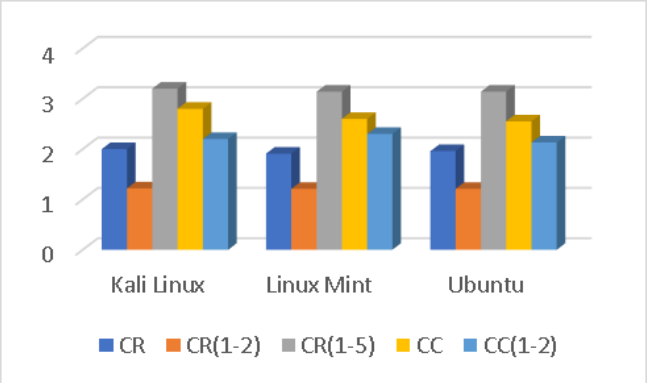

Fig 5- ELT Algorithm implemented across various Platforms

Figure 5 contains a bar graph containing the run-time of various operating systems running ELT Algorithm Include a note with your final paper indicating that you request color printing. According to the first chart, for Kali Linux an average count of run time for various classification of DAG were run and respective time was being calculated. The other bar graph contains the bar chart of other two operating systems for Linux Mint and Ubuntu. . Here the execution time was given in unit times.

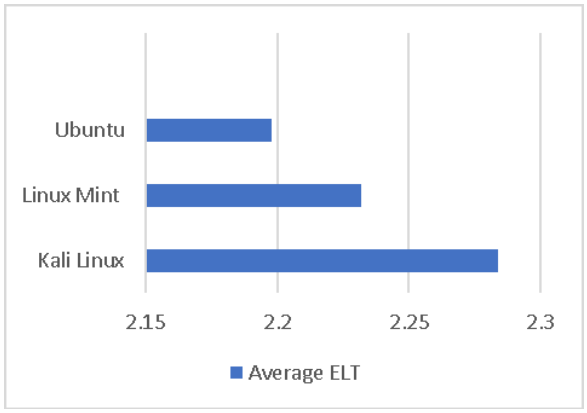

Fig 6- Average Execution time of ELT

The figure 6, average execution time of the algorithm was calculated. Here from the figure we can see, Kali Linux having the highest execution time as a result is the least favorable platform for running the ELT algorithm. Then the least one is found to be Ubuntu which required the least execution time in time units. This is because Ubuntu provides the best support for development and allows best support for the compiler used. On the other hand, Ubuntu provides the fastest execution of the executable as the time required mapping the memory into multiple files is faster than Kali and Linux Mint where both are heavier compared to Ubuntu which is the most lightweight among these three.

Conclusion

This research provides information on the run time of the ELT algorithm on different Linux based OS. The algorithm was implemented using C++ and then it had been run on Kali Linux, Linux Mint and Ubuntu. The algorithm was implemented on each of the operating systems, where it was found the runtime of Linux mint was 2.284, kali Linux was 2.232 and Ubuntu was 2.198-unit times. From the above, we can conclude Ubuntu as the most efficient Linux platform for the implementation of the algorithm. However, the current algorithm is optimized for use in tasks that involves graph. It will be very useful if it can be applied on all kinds of tasks of a computing system which can be done in future.

Acknowledgment

We would like to thank deeply to Mr. Mohammad Marufuzzaman, Post-Doctoral Researcher, The National Energy University, Malaysia for his utmost support and guidance throughout this research.

References

[1] https://en.wikipedia.org/wiki/Distributed_shared_memory
[2] EsmaInsafDjebbar and GhalemBelalem, “Tasks Scheduling and Resource Allocation for High Data Management in Scientific Cloud Computing Environment”, Springer International Publishing AG 2016 S. Boumerdassi et al. (Eds.): MSPN 2016, LNCS 10026, pp. 16–27, 2016. DOI: 10.1007/978-3-319-50463-6_2
[3] uccar, I , Solar, M., Kri, K, Parada, V, 2006, in IFIP International Federation for Information Processing, Volume 218, “Professional Practice in Artificial Intelligence”, eds. J. Debenham, (Boston: Springer), pp. 313-321.
[4] M. Solar and M. Feeley, “A Scheduling Algorithm considering Latency Time on a shared Memory Machine”, 16th IFIP World Computer Congress 2000, Beijing, China (Aug., 2000).
[5] Michael Stumm and Sognian Zhou, “Algorithms Implementing Distributed Memory”, IEEE Computer, 23(5), Msy 1999, pp – 54-64
[6] Martin Josef Geiger, SanjaPetrovic, “An Interactive Multicriteria Optimisation Approach To Scheduling, Automated Scheduling”, Optimisation& Planning Research Group
[7] B. Kruatrachue and T. Lewis, “Duplication Scheduling Heuristics: A New PrecedenceTask Scheduler for Parallel Processor Systems”, Technical Report, Oregon State University (1987).
[8] K Ramamritham, JA Stankovic, “Scheduling algorithms and operating systems support for real-time systems”, Proceedings of the IEEE, 1994
[9] ArnavWadhonkar, Deepti Theng, “A survey on different scheduling algorithms in cloud computing,”, 2016 2nd International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB)
[10] Pete Keleher, Alan L. Cox, Sandhya Dwarkadas and Willy Zwaenepoel, “Distributed Shared Memory on Standard Workstations and Operating Systems”, Department of Computer Science, Rice University, Houston, TX 77251-1892