International Journal of scientific research and management (IJSRM)
\||Volume||3||Issue||6||Pages|| 3134-3138||2015||
Website: www.ijsrm.in ISSN (e): 2321-3418

Real-Time Monitoring Framework for Parallel Processes

Swagatika Kar

Department of Computer Science, Berhampur University
Berhampur-760007, India
swagatikakar31@gmail.com

Abstract: *This paper describes the development of a Real-Time monitoring framework for parallel processes. There are various parallelism techniques in parallel execution of processes such as Bit-Level parallelism, Instruction-Level parallelism, task parallelism. Besides these techniques, different hardware architecture of machines for processing like pipelining of processes, superscalar execution, data dependency, resource dependency are there. By using any of these parallelism techniques or hardware architecture, Real-Time parallel processing applications can be developed. Our objective is to develop a data- driven framework for these Real-Time parallel processing applications. The framework monitors entire lifecycle of processes executing in that system and gather granular level data to conclude entire system information. This information can be used to reduce the overall execution time, chances of deadlocks, production cost along with improvement in efficiency, consistency, resource utilization. Soft sensors record the entire details of process during its execution like execution time, waiting time, delay etc. throughout life of processes.*

Keywords: Real-Time monitoring, Real-Time Execution Analyzer framework, Real-Time parallel processing**.**

1. Introduction

Parallel processing is the execution of a process simultaneously on multiple processors by splitting it into smaller pieces. It is widely used to reduce the overall execution time for complex and large-scale tasks. Multiprocessor systems consist of processors with aggressive branch predictors execute many memory references and resources distributed over network locations that might turn out to be on a mispredicted branch path. Thus in this current scenario of non-linear and dynamic process environments, it is very difficult to achieve optimal performance in parallel processing systems. This can concern in many industries especially in telecommunication. For many applications it is imperative that parallel execution is maintained within strict limits, as improved consistency will result reduced production cost and efficient performance.

Applications of Tele-communication and Messaging services use different parallelism techniques over variety of hardware architectures. Common decomposition and common mapping methods are efficient techniques to achieve parallel processing, still fault exists with the resource utilization and data dependency of tasks executing in parallel. Common decomposition method decomposes a process into small pieces known as tasks. These tasks are executed independently and mapped together in common mapping methods to produce the expected outcome.

In several distributed parallel processing systems like ABS and ABROSE, CORBA was used as connectivity layer [1]. CORBA facilitates the inter-module communication management on different servers. Invocation of CORBA is like non-blocking calls, so there is no difficulty in case of asynchronous or parallel communications. Utilization of CORBA is responsible for decrease in execution speed as compared to execute the entire module in the same JVM. Our framework is designed in Java platform and Smart-EC [1] is considered as the best choice for the communication architecture. Smart-EC also provides communication infrastructure for inter or intra communication between different modules of system. It is purely implemented on J2EE technology, which is based on Java Message Service (JMS) for synchronous and asynchronous/parallel communication. JMS is an asynchronous communication API developed for Java application and lately included with J2EE versions. Execution of entire module on same JVM makes the communication faster and improves the efficiency of system. The multi-tier architecture of J2EE technology also makes the system robust.

2. Issues in Parallel Processing Systems

There has been extensive amount of research carried out on parallel processing systems since many years [3][4][5]. Parallel processing has become very important while researching high-performance solutions. Therefore, some common issues are to be monitored continuously to improve the performance and efficiency of any parallel processing system.

Memory Access: In recent years, the speed of processors has been increased much faster than the DRAM speeds. As the clock rates of high-end processors have increased at roughly 40% per year over the past decade, DRAM access have only improved at the rate of roughly 10% per year. It is expected that the processors speed will be double in each upcoming 18 months which is supposed to continue for the next 10 years. This will result significant increase in memory latency and leads to heighten attention on program memory performance, due to potential for access over a network to remote memory.

Data and Control dependency: In a parallel processing system programs are divided into smaller units and these smaller units are executed on different processors. The execution of one instruction may depend on execution of another instruction(s) in terms of data or control. This may either limit the performance in terms of efficiency and execution time or may lead to possibility of a hazard like detention of a process or deadlock.

Inter-process communication: Inter-process communication is one of the major issues in architecture of parallel processing system. It's due to data dependency or network configuration. If the processing is not synchronized properly, two processes executing concurrently reading and writing data on shared memory may lead to incorrect data processing.

3. Implementation of Monitoring Framework

In this paper we are introducing a new framework RTEA (Real-Time Execution Analyzer) and implementing it on java platform. We can apply multiple soft sensors on top of any real time parallel processing application to monitor entire execution cycle. These soft sensors monitor execution of instructions on each clock cycle of processors. To identify the exact point of issues in Real-Time parallel processing applications, we apply the processing of RTEA framework at both grain level of parallelism.

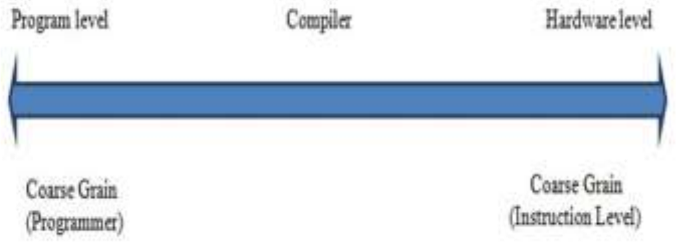


Figure-1 Process Granularity

The framework collects all data during execution of a process through soft sensors. Finally, these collected data are analyzed using RTEA framework. The analysis mechanism involves "Genetic Programming". It produces the exact processing data and analyzes it to find fault or inefficiency in existing parallel real time application. It also finds the solution for these issues, which leads to improve the performance and efficiency of application.

4. Processing of Monitoring Framework

4.1 Importance of continuous execution monitoring

Continuous execution monitoring of all processes running parallely in a real time application is essential. RTEA framework injects soft sensors to the application and continuously monitors the execution of all processes throughout their life. The details of each individual process like waiting time, number of sub-processes, execution time, occurrence of faults during execution etc. are recorded by soft sensors and stored in database. Analyzer tool uses these data to produce deficiencies in the existing system.

Benefits of continuous monitoring of execution of processes are discussed below:-

By recording execution time of all processes and after analyzing them, we can find out the reasons behind waiting time, latency time, turnaround time and also detention of a process from execution. Due to these reasons the algorithm or the application hardware architecture is lagging behind. We can rectify and enhance the algorithm or the hardware/network architecture to reduce the total execution time of each process and also enhance the throughput of the application by reducing number of fault occurrence.

RTEA framework calculates the execution time of processes. There are number of ways to calculate the execution time like using date command or time command, counting clock cycle of processors, using Timer/Counter chip etc. Here we are using counting clock cycle method to calculate the entire execution time. Soft sensors invoke clock( ) command to count the number of clock cycle(s) for each task individually and then these clock cycle(s) for all tasks are added to get the total clock cycle(s). We are using Cycles Per Element (CPE) measure for programs which requires an accuracy of 0.1% for a procedure having a CPE of 10. A processor clock controls the execution of instruction while a precision oscillator regulates it.

The processor hardware works at a microscopic time scale, where instruction having durations of a few nanoseconds (ns). The system must deals with a macroscopic time scale, with events having durations of a few milliseconds (ms).

The total execution time of a task is divided into two categories i.e., Active (executing its instructions) and Inactive (waiting for the instruction to be scheduled for execution). Soft sensors trigger a program in the RTEA framework. It traces the inactive time, finds the reason behind it and stores all information.

The analyzer tool fetches all data stored by the framework and extracts the reasons behind overall inactive time of processors. It produces information about drawbacks in system development and configuration architecture. This can help system designers and algorithm developers of the concern system to find the feasible solution in order to reduce the overall inactive time of tasks (It may be from software design or hardware architecture). This can improve the performance of real-time application.

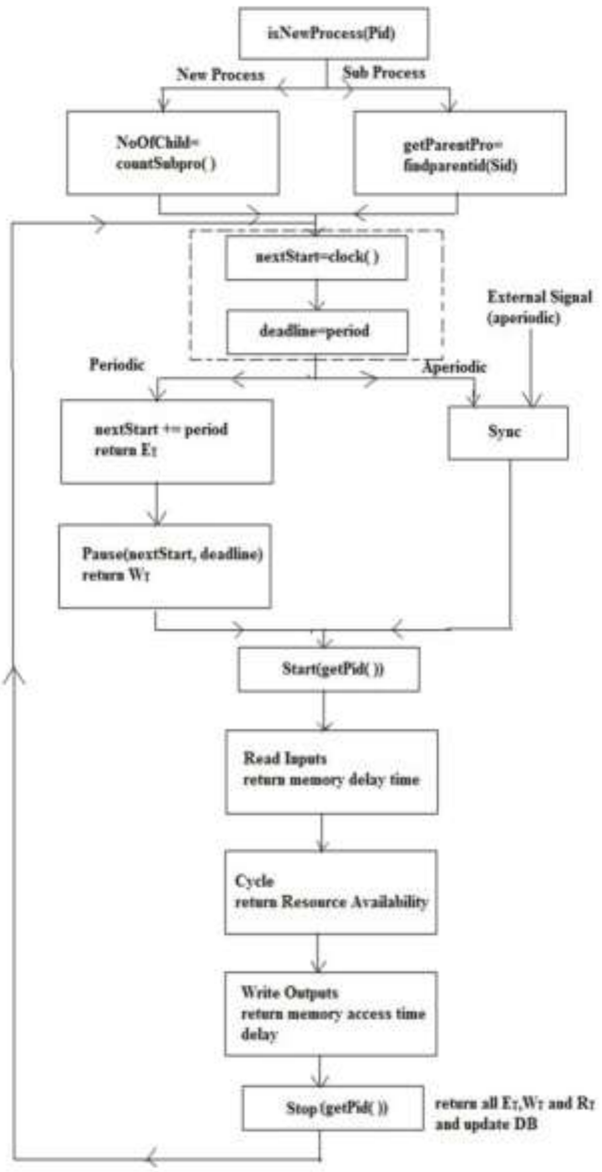
Measuring total execution time of code is helpful for optimization of code, but it is not sufficient enough to analyze real time system performance. Rather, measuring overall execution time is an effective step for estimating and analyzing a feasible solution to enhance the system performance. Figure-2 and pseudocode define the basic structure to calculate the total execution time of synchronous and asynchronous programming.

```
newprocessflag=isNewProcess(getPid);
nextStart = clock( );
deadline = period;

while(task-> state == ON){
       if(tasktype == PERIODIC){
       nextstart += task-> period;
       Pause(nextStart,period);
       return $E_T$ and $W_T$;
       }
else{
  task->func->sync(task->local);
       return $E_T$ and $W_T$;
       }
START(task->id)
readInputs( );
task->function->cycle (task->local);
writeOutouts( );
STOP(task->id);
Return $E_T$,$W_T$ and $R_T$;
}
```

[Pseudocode for framework execution time calculator]

Figure-2 Framework execution time calculator


isNewProcess(Pid)
New Process
Sub Process
NoOfChild= countSubpro( )
getParentPro= findparentid(Sid)
nextStart=clock( )
deadline=period
External Signal (aperiodic)
Periodic
Aperiodic
nextStart += period return E_T
Sync
Pause(nextStart, deadline) return W_T
Start(getPid( ))
Read Inputs return memory delay time
Cycle return Resource Availability
Write Outputs return memory access time delay
Stop (getPid( ))
return all E_T,W_T and R_T and update DB

4.2 Importance of granular level of data to be recorded

In parallel computing, granularity is the ratio of computation to the amount of communication [2]. Fine-grained parallelism means the individual units of tasks are relatively smaller in terms of number of instruction and execution time. So, data transfer between processors occurs frequently in amounts of one or a few memory words. In case of coarse-grained, data communication between processors occurs infrequently, after large amount of computation. Fine-grained increases the potential of parallelism and speedup, but it also increases the overheads of synchronization and communication.

We need exact level of granularity in our algorithm to reduce the overhead up to maximum without compromising with speedups. The hardware architecture in terms of memory units and network communication should be faster. This reduces the response time and affects the total execution time of tasks. So, here we focus on coarse grain level of granularity, which restricts to make smaller units of tasks and task dependencies unnecessarily.

RTEA framework calculates the number of units in which tasks are divided. It stores all information about the instruction execution. It fetches dependency details of the executing instruction with others instruction and generates dependency graphs. Our framework reads all data, then analyzes them and generates a report with collected details. It informs the administrator about the particular points of execution, which are lagging behind and responsible for low performance of the system. The report contains a detail graph of all instructions within processes. The graph contains processes and processes are divided into tasks. Each task contains instructions, sequence of instructions inside a task and the total time for execution. With the help of this graph, programmers and architects can easily identify which instruction is taking more time than the expected and reason behind that. After gathering all reasons they can accordingly modify their algorithm to achieve higher performance and accuracy.

```
getAllChild(getPid())
{
  for(inti=1;i<=NoOfSubProcess;i++){

  if(i==1)
       break;
       elseif
(checkdependency($P_{11}$,$P_{1i}$))
       return true;
       else
       return false;
       }
}
```

[Pseudocode to check dependencies]

4.3 Monitoring of utilization of processors and other resources

In hardware architecture, processors and the other resources (memory units, network communication design etc.) are main units on which execution and performance depends. Also the hardware units are expensive. So we should be careful while selecting and implementing the required hardware. The algorithm should utilize the resources optimally to achieve the expected performance. This RTEA framework also helps to verify the usage and performance of the hardware units and processor, and generates all details which helps designer to modify the design.

These soft sensors trigger over all the resources used in the system. These trigger record all activities of each individual resources like idle time, load on resources, execution time of each instruction and the reasons behind it. The database stores these data in tabular format. Later analyzer tool analyzes these data and generates a report. This report contains information about utilization of all resources in the system.

Soft sensors are also responsible for tracking failed and detained processes. By analyzing the stored data we can find the exact reasons behind the detention or failure. It helps the designers and programmers to enhance the existing algorithm or the hardware/network architecture of the system. This reduces the number of fault occurrence and results in higher performance.

4.4 Importance of monitoring of communication network

In parallel processing system, communication network plays a vital role in execution of all processes. There are three major parts of network communication.

Inter-process communication:

Inter-process communication is the communication between different processes. In other words, Inter-process communication deals with the data transfer between tasks or instructions of different processes executing in parallel. RTEA framework triggers the soft sensors to all instructions executing in parallel and calculates dependencies between them. It also finds the sequence of execution and stores these data in database. The analyzer tool analyzes these data and generates report. From this report programmers and hardware designers can identify the reasons for delay in execution. Programmers can identify which instruction is taking more time than expected and why. Then they can modify the system to reduce unwanted or inefficient dependency.

Data communication network:

In parallel processing system there are shared memory units and different processors access them simultaneously. Processors are connected to shared memory units by data buses which creates data communication network. RTEA framework monitors them and generates report. Then design architects can identify the memory units taking long time to respond. They can also design new memory architecture as required or can implement some faster accessible memory units like virtual memory, cache etc. So, it can reduce the overall memory access cycle, increase overall performance and also can reduce the occurrence of fault due to memory model.

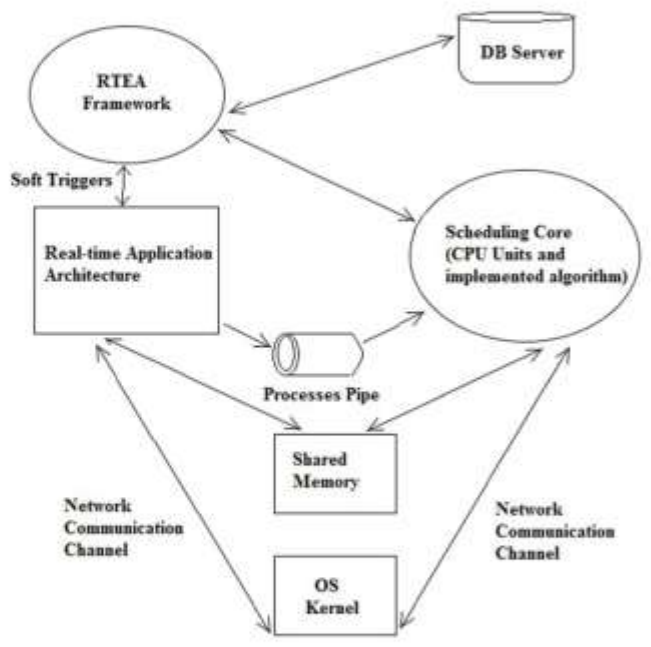
Network communication:

Implementation of network communication is necessary due to usage of multiple shared resources (devices used for processing), requirement of current online data over internet like weather condition, traffic condition etc. RTEA framework triggers its commands using soft sensors to fetch all details of the time spent by processors over network communication channel for each instruction. Then it generates detail report. Using this report network designers can modify network designs, can use faster buses to reduce network traffic and the response time to the processors. The overall modifications can improve the performance of the system with reduced number of faults.

5. Analyses of Data and Proposed Solution to Improve Performance

We applied RTEA framework over multiple existing real time parallel processing systems like intercommunication and notification system in telecommunication, brokering and negotiation system. These are based on windows sever machine where different modules of application are deployed on individual nodes and found significant result. The changes made in algorithm and the hardware architecture on the basis of report generated with the help of this framework results in higher performance of system with reduced cost, reduced excess implementation of resources (Resource utilization) and reduced number of faults.

Here we are presenting the analysis and data for Virtual Machine scheduling as a proof, where we applied this framework.



RTEA Framework
DB Server
Soft Triggers
Real-time Application Architecture
Scheduling Core (CPU Units and implemented algorithm)
Processes Pipe
Shared Memory
Network Communication Channel
OS Kernel
Network Communication Channel

Figure-3 System Architecture with Real-Time Application and RTEA Framework

As in Figure-3, RTEA framework was implemented on top of a Real-Time application. There were 4 dual-core processors used. Assume that, four tasks were triggered to the Real-Time application. As per the algorithm implemented each process is divided into 11 tasks and their dependencies are given in Figure-4.

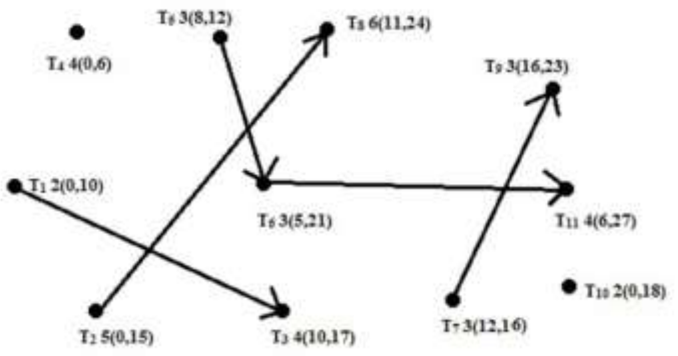


Figure-4 Tasks dependency graph

The RTEA framework tracked each execution cycle of the processors for all the processes executing in the system and provided all information for analysis. We have illustrated results below for one process in execution. It tracked all information about each individual instruction of process in each processor clock cycle.

Table-1 Tasks details by RTEA framework, where: e_i:- execution time, r_i:- release time, d_i:- deadline

	e_i	r_i	d_i
T_1	2	0	10
T_2	5	0	15
T_3	4	10	17
T_4	4	0	6
T_5	3	8	12
T_6	3	5	21
T_7	3	12	16
T_8	6	11	24
T_9	3	16	23
T_{10}	2	0	18
T_{11}	4	6	27

In Figure-4, one process was divided into 11 tasks. Individual weights and CPU times were assigned by the scheduler/algorithm. There were 4 processes executing simultaneously. So, total 44 tasks were scheduled by 4 dual-core processors i.e., 8 CPUs in parallel. As per scenarios we identified that each task was assigned to two CPU units to execute its instructions at a given time. So, here we are taking one process and two CPU units into account to calculate all the data and illustrate the outputs.

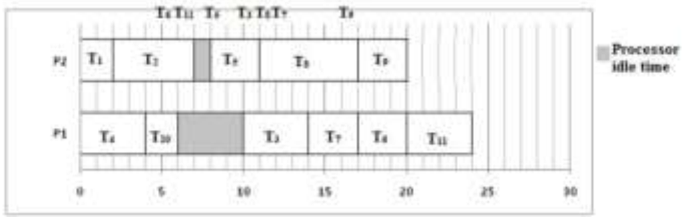


Figure-5 Tasks scheduling diagram

From Figure-5,
Total processor idle time=5
Total waiting time of tasks=34
Total execution cycle=39
Total execution time=78 clock cycles
For this above execution cycle, RTEA framework stored these information in database and generated a report containing these details. It also generated reports for processor time spent for communication channels, memory units and other resources used in application. Now from Figure-5, we can see that sometimes processor was idle because there was no available task for execution. Sometimes tasks were waiting for other tasks to be executed first because they were dependent on those other tasks. After viewing these results, the process slicing algorithm was modified to reduce the number of sliced tasks and their dependencies. Then RTEA framework was again used to track each execution cycle of the processors for all the processes executing in the system and provided all information for analysis.

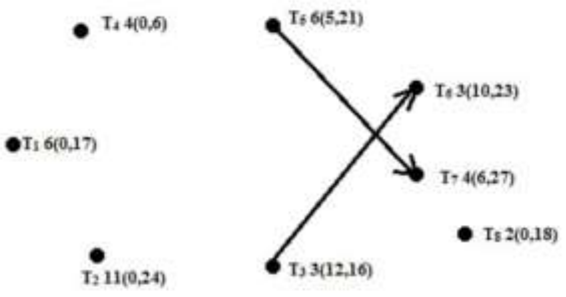


Figure-6 Tasks dependency graph

Table-2 Tasks details by RTEA framework

	e_i	r_i	d_i
T_1	6	0	17
T_2	11	0	24
T_3	3	12	16
T_4	4	0	6
T_5	6	5	21
T_6	3	10	23
T_7	4	6	27
T_8	2	10	18

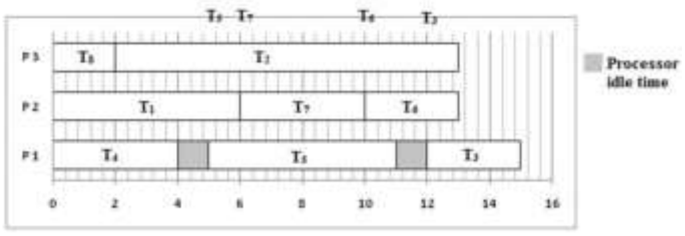


Figure-7 Tasks scheduling diagram

From Figure-7,
Total processor idle time=2
Total waiting time of tasks=0
Total execution cycle=39
Total execution time=42 clock cycles
Now, we can see that the dependencies of tasks are decreased so as the total execution time. By analyzing these results designers and developers can redesign their system. The changes made on the basis of reports generated by RTEA framework enhanced the overall performance of the system. Similarly, we can also track the time consumed by other resources in the system.

6. Conclusion

In this paper we proposed a framework named Real-Time Execution analyzer for monitoring of Real-Time parallel processing applications. This framework can be implemented for any Real-Time parallel processing system. It can monitor execution of all processes which enhance the overall performance with minimal cost. This RTEA framework was implemented over a Virtual Machine Scheduler in this work and significant results were found.

References

[1] Valera F., Solages A., Bellido V. and Bellido L. :Parallelism and messaging services in a J2EE-based e-commerce brokering platform .Experiences in European projects. In: International Conference on Electronic Commerce and Rearch-5
[2] https://en.wikipedia.org/wiki/Granularity
[3] Belikov E., Deligiannis P. amd Totoo P. A survey of High-Level parallel programming models. In:Technical Report HW-MACS-TR-0103, Heriot-Watt University, Dec 16, 2013
[4] Lakshmanan K., Kato S., and Rajkumar R. :Scheduling parallel Real-Time tasks on multicore processors :In Proc. RTSS'10, 2010, pp. 259–268
[5] Saifullah A., Agrawal K., Lu C., and Gill C. :Multi-core Real-Time scheduling for generalized parallel task models :In Proc. RTSS'11,2011, pp. 217–226

Author Profile

Swagatika Kar received the B.Tech. (Computer Science & Engineering) degree from Synergy Institute of Engineering and Technology and M.Tech. (Computer Science) degree from Berhampur University in the years 2010 and 2014, respectively. Her area of research involves Software Engineering, Data Mining, Real-Time Systems, Cryptography and Service-Oriented Architectures.