Performance Analysis Of Concurrent Tasks Scheduling Schemes In A Heterogeneous Distributed Computing System

Bibhudatta Sahoo
Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Orissa, INDIA
bdsahu@nitrkl.ac.in
Aser Avinash Ekka
Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Orissa, INDIA

Abstract

Performance of distributed systems can be improved from scheduling of tasks aspect. A good scheduling algorithm can enhance the performance of the distributed system significantly. In this paper we have compared the performance of *batch mode* and *immediate mode* schedulers in heterogeneous distributed computing environment. An immediate mode scheduler only considers a single task for scheduling on a FCFS (first come, first served) basis while a batch mode scheduler considers a number of tasks at once for scheduling. In particular we have used two immediate mode scheduler: *(i) the earliest first (EF) algorithm* and (ii) *the lightest loaded (LL),* and two batch mode heuristic scheduler *(i) the max-min (MX) scheduler* and *(ii) min-min (MM) scheduler.* The main aim of max-*min (MX) scheduler* is to have the largest tasks scheduled as early as possible, with smaller tasks at the end filling in the gaps. The min-min (MM) scheduler is similar to the MX scheduler, except tasks are sorted in ascending order according to size. We have simulated the scheduler behavior with our simulator developed using Matlab, where each task is with the *expected execution time* and *expected completion time* on a particular machine. This findings are used to design an adaptive dynamic scheduler that selects the best strategy depending on load at a particular time frame. The results are also useful in deciding the effective group size of a processor pool (cluster) for the HDCS, which can be remodeled as a tree of resource clusters that are geographically distributed. We have also outline the proposed scheduler framework that uses (i) a global scheduler, responsible for determining where to send task submitted to it, a local scheduler, responsible for determining the order in which tasks are executed at that particular processor pool.

1. Introduction

Heterogeneous distributed computing system (HDCS) utilizes a distributed suite of different high-performance machines, interconnected with high-speed links, to perform different computationally intensive applications that have diverse

computational requirements. Distributed computing provides the capability for the utilization of remote computing resources and allows for increased levels of flexibility, reliability, and modularity. In heterogeneous distributed computing system the computational power of the computing entities are possibly different for each processor [1,3,4,11]. A large heterogeneous distributed computing system (HDCS) consists of potentially millions of heterogeneous computing nodes connected by the global Internet. The applicability and strength of HDCS are derived from their ability to meet computing needs to appropriate resources[2,3].

Resource management sub systems of the HDCS are designated to schedule the execution of the tasks that arrive for the service. HDCS environments are well suited to meet the computational demands of large, diverse groups of tasks. The problem of optimally mapping (defined as matching and scheduling) these tasks onto the machines of a distributed HC environment has been shown, in general, to be NP-complete, requiring the development of heuristic techniques to obtain an acceptable solution under certain QoS [12,13].

We have considered the HDCS where, the real time tasks are assumed to be independent, i.e., no communications between the tasks are needed. The individual users of the systems are independently submitting their jobs to the central scheduler. The central scheduler operates using a dynamic scheme, because the arrival times of the tasks may be random and some machines in the suite may go off-line and new machines may come on-line. The performance of dynamic mapping heuristics schemes has been investigated in this study are non-preemptive and assume that the tasks have no deadlines or priorities associated with them. Simulation studies are performed to compare two immediate mode scheduler: *(i) the earliest first (EF) algorithm* and (ii) *the lightest loaded (LL),* and two batch mode heuristic scheduler *(i) the max-min (MX) scheduler* and *(ii) min-min (MM) scheduler* with three different task pattern[5].

The rest of the paper is organized as follows. The next section discusses Heterogeneous distributed computing system (HDCS) structure. *Section 3* describes the different tasks Scheduling Schemes used to schedule the real time task, We have simulated the scheduler behavior with our simulator developed using Matlab, where each task is with the expected execution time e_{ij} and expected completion time c_{ij}, of task t_i on machine m_j. The results of the simulation with different scheduler

on various task patterns are presented in *Section 4.* Finally, conclusions and directions for future research are discussed in *Section 5.*

2. Heterogeneous distributed computing system

A Heterogeneous distributed computing system consists of a set of *N* heterogeneous computers interconnected together via a network. Each computer has some computational facilities and a local memory. A HDCS consists of three types of nodes: distributor nodes for distributing pieces of a distributed computation, client nodes for executing these pieces and reporting results back to a distributor node, and portal nodes for serving as central sites where client nodes can be directed to distributor nodes. The processors of the distributed system are heterogeneous and the availability of each processor can vary over time (processors are not dedicated can may have other tasks that partially use their resources)[1,8,9]. A simple heterogeneous distributed computing system is show in figure 1.

We consider a heterogeneous distributed computing system (HDCS) consists of a set Ω of n Nodes (uniquely addressable computing entity)$\{P_1, P_2, \ldots, P_n\}$, $P_i=(\Delta_i, \varepsilon_i)$, where Δ_i, is the set of tasks in the queue of P_i , ε_i is the fixed execution rate. Each processor was assumed to have different execution rate measured in MFLOPS/s. The expected execution time e_{ij} of task t_i on machine m_j is defined as the amount of time taken by m_j to execute t_i given m_j has no load when t_i is assigned. The expected completion time c_{ij} of task t_i on machine m_j is defined as the wall-clock time at which m_j completes t_i (after having finished any previously assigned tasks). Let n be the total number of machines(nodes) in the HDCS suite. Let Δ_i be the set containing the tasks that will be used in a given test set for evaluating heuristics in the study. Let the arrival time of the task t_i be a_i , and let the time *ti* begins execution be *bi* . Then the completion time of the task can be computed as, $c_j = b_i + e_{ij}$. Let c_i be the completion time for task t_i , and it is equal to c_{ij} where machine m_j is assigned to execute task t_i. The makespan for the complete schedule is then defined as max $t_i \in \Delta_i$, [16]. Makespan is a measure of the throughput of the HDCS[18].

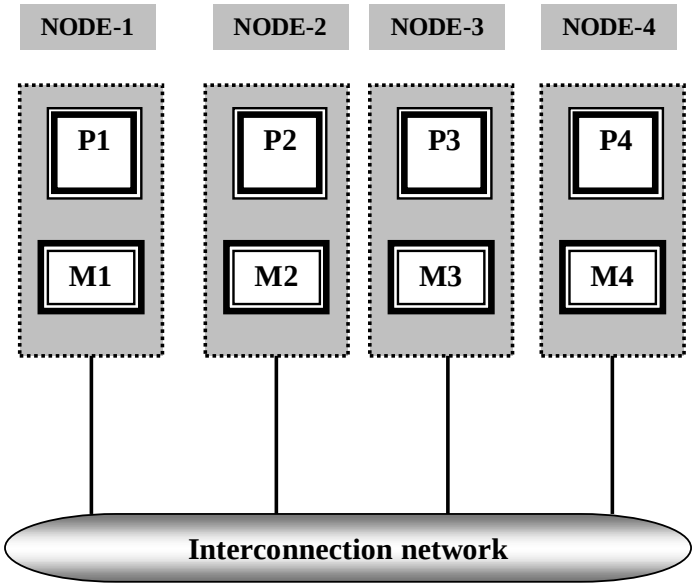

Figure:1 Distributed Computing System

The *resource manager* schedules the tasks in a distributed system to make use of the system resources in such a manner that resource usage; response time, network congestion, and scheduling overhead are optimized. There are number of techniques and methodologies for scheduling processes of a distributed system[2,4,6].

The dynamic mapping heuristics investigated in this paper are non-preemptive and assume that the tasks have no deadlines or priorities associated with them. The mapping heuristics can be grouped into two categories, immediate mode and batch mode heuristics. In the immediate mode, a task is mapped onto a machine as soon as it arrives at the central scheduler. In the batch mode, tasks are not mapped onto the machines as they arrive; instead they are collected into a set that is examined for mapping at prescheduled times called mapping events. The independent set of tasks, which are considered for mapping at the mapping events is called a meta-task. A meta-task can include newly arrived tasks (i.e., the ones arriving after the last mapping event) and the ones that were mapped in earlier mapping events but did not begin execution. While immediate mode heuristics consider a task for mapping only once, batch mode heuristics consider a task for mapping at each mapping event until the task begins execution. The trade-offs among and between immediate mode and batch mode heuristics are studied experimentally with our simulator developed using Matlab. The next section discusses in detail the task scheduling schemes use for this study.

3. Tasks Scheduling Schemes

The real-time scheduling based on the timing constraints of the tasks in a real-time environment. There are two kinds of tasks: ***periodic*** and ***aperiodic*** (***sporadic***). The periodic tasks must run repeatedly, and within fixed times. The aperiodic tasks run sporadically, and only once when we invoke them. The real-time scheduling is very complex. The tasks are to execute in HDCS within their timing constraints, responding to the high critical tasks first. The scheduling algorithms of many operating systems used at present for real-time processing are simple extensions of those used in Time-Sharing systems. Most of them use priority algorithms, letting the programmer to adjust the task priorities to fulfill the timing constraints. Moreover, the designer must map many conflicting considerations (timing constraints, criticality, task dependencies and others) in only one number: the task priority. The only way to guarantee predictable behavior is through exhaustive testing. A real-time scheduling algorithm must insure [16]:

1. ***Predictable*** response time of tasks.
2. High degree of resource employment (***schedulability***), while keeping predictable responses.
3. ***Stability*** under transient overloads. In these cases, the scheduler must guarantee the response time of a selected group of critical tasks.

Most of the real-time schedulers also use priorities' schemes (static or dynamic). Instead, the dynamic approach allows the task priority to change during the program execution. It is also important to the scheduler to be preemptive. A non-preemptive scheduler could lead to run a low priority task while a high priority task is waiting. For a given scheduling algorithm, an optimal task assignment algorithm achieves a feasible schedule for each processor with the least number of processors. A schedule, in which all real-time tasks are executed within their deadlines and all the other constraints, if any, are met, is called a *feasible schedule*.

In the immediate mode heuristics, each task is considered only once for matching and scheduling, i.e., the mapping is not changed once it is computed. When the arrival rate is low enough, machines may be ready to execute a task as soon as it arrives at the ready queue. Therefore, it may be beneficial to use the scheduler in the immediate mode so that a task need not wait until the next mapping event to begin its execution.

As described in section 2, *immediate mode*, the *scheduler* assigns a task to a machine as soon as the task arrives at the mapper, and in batch mode a set of independent tasks that need to be mapped at a mapping event is called a meta-task[6,18]. (In some systems, the term meta-task is defined in a way that allows inter-task dependencies.) In batch mode, for the ith mapping event, the meta-task(group) *Mi* is mapped at time τ_i , where $i \geq 0$. The initial meta-task, M_0, consists of all the tasks that arrived prior to time τ_0. The meta-task, M_k, for $k>0$, consists of tasks that arrived after the last mapping event and the tasks that had been mapped but had not started executing, i.e., $M_k = \{ t_j \mid \tau_{k-1} \leq a_j < \tau_k \} \cup \{ t_j \mid a_j < \tau_{k-1}, b_j > \tau_k \}$

In batch mode, the scheduler considers a meta-task for matching and scheduling at each mapping event. This enables the mapping heuristics to possibly make better decisions than immediate mode heuristics. This is because the batch heuristics have the resource requirement information for a whole meta-task and know about the actual execution times of a larger number of tasks (as more tasks might complete while waiting for the mapping event). When the task arrival rate is high, there will be a sufficient number of tasks to keep the machines busy in between the mapping events and while a mapping is being computed. (It is, however, assumed in this study that the running time of each mapping heuristic is negligibly small as compared to the average task execution time.)

Both immediate mode and batch mode heuristics assume that estimates of expected task execution times on each machine in the HC suite are known. The assumption that these estimated expected times are known is commonly made when studying mapping heuristics for HC systems (e.g., [14, 15, 17]).

Two batch mode heuristics are described here: (i) the Min-min heuristic, and (ii) the Max-min heuristic. In the batch mode heuristics, meta-tasks are mapped after predefined intervals.

The ***Min-min heuristic*** is a two-step task scheduler. First, select a “best” (with minimum completion time) machine for each task. Second, from all tasks, send the one with minimum completion time for execution. The idea behind Min-min is to send a task to the machine, which is available earliest and executes the task fastest.

```
1   while there are tasks to schedule
2       for all task i to schedule
3           for all host j
4               Compute CTi,j = CT(task i, host j)
5           end for
6           Compute metrici = f1(CTi,1, CTi,2, ……)
7       end for
8       Select best metric match (m,n) = f2(metric1,metric2,……)
9       Compute minimum CTm,n
10      Schedule task m on n
11  end while
```

ALGORITHM-1 General Scheduling

The Max-min heuristic takes the same first step as Min-min but send the task with maximum completion time for execution. This strategy is useful in a situation where completion time for tasks varies significantly. Using this heuristic, the tasks with long completion time are scheduled first on the best available machines and executed in parallel with other tasks. This leads to better load-balancing and better total execution time.

The general scheduling algorithm [*Algorithm-1*], iteratively assign tasks to processors by considering tasks not yet scheduled by computing their expected Minimum Completion Time (MCTs). For each task (line 2), this is done by tentatively scheduling it to each host (line3), estimating the task's completion time on it (line 4). For each task, a metric function "f1" is computed over all the hosts (line 6). Afterwards, the task/host pair with the best metric match (m,n) is selected using selection function "f2" (line 8). We then compute the minimum completion time of this task/host pair (line 9) and assign the task m to the host n (line 10). The process is then repeated until all tasks have been scheduled (line 1 and line 11).

The Min-Min, Max-Min heuristic define "f1" as the minimum completion time, that is, for task i, they select the minimum completion time over all the hosts. However, in function "f2", the Min-Min selects the minimum completion time over all tasks (the minimum $metric_i$), whereas the Max-Min selects the maximum completion time over all tasks (the maximum $metric_i$).

The asymptotic complexity for the max-min and min-min algorithms is $O(\mu\tau^2)$, where μ is the number of machines in the heterogeneous computing and τ is the number of tasks to execute. The performance of the different scheduling model has been analyzed and compared for real time tasks with different traffic patterns.

4. Simulation Procedure and Outcomes

The mappings are simulated using a discrete event simulator developed by us using Matlab 6.0. A Poisson, uniform and normal distribution process, models the task arrivals. The simulator contains an ETC (expected time to compute) matrix that contains the expected execution times of a task on all machines, for all the tasks that can arrive for service. The ETC matrix entries used in the simulation studies represent the e_{ij} values (in seconds) that the heuristic would use in its operation. The results of the simulation with different scheduler on various task patterns are shown in figure 1-4. Each data point in the comparison figure is an average over 100 trials, where for each trial the simulated actual task execution times are chosen independently. It has been observed that irrespective of task arrival pattern ***max-min*** and ***min-min*** algorithm are found to be efficient. In most of the HDCS are very often design to performs tasks that are periodic in nature. We have used three different approach *(i) max-min, (ii) FCFS* and *(iii) randomized* technique to schedule periodic tasks with arrival rate as Poisson, The simulation results in figure 4 shows that, the number of processors that can be group in a HDCS so that we can best utilize them to get maximum speedup for the periodic task[7].

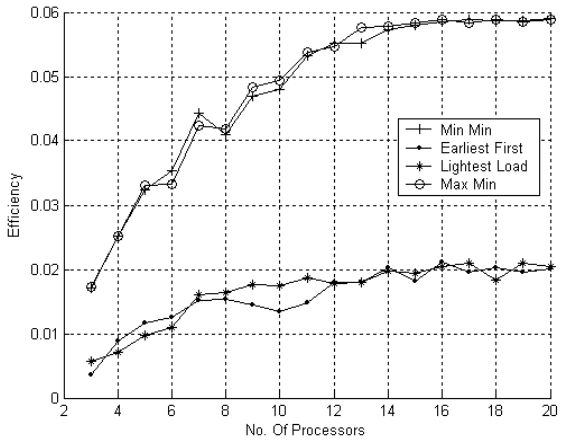

Figure 1. Task size over Poisson distribution

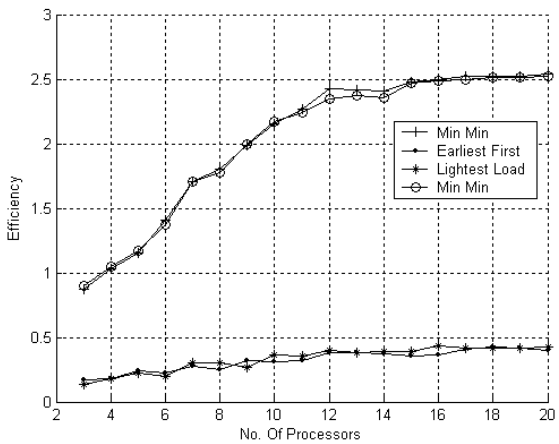

Figure 2. Task size over Uniform distribution

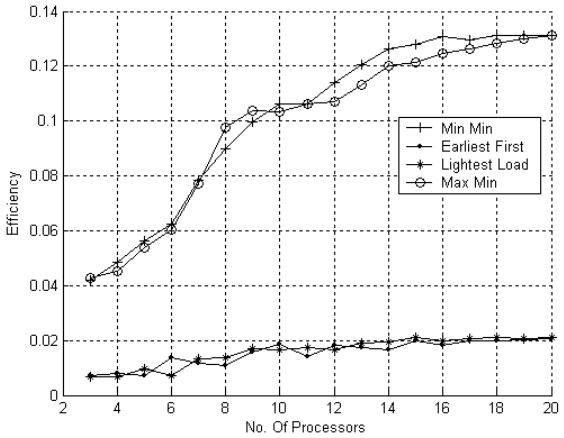

Figure 3. Task size over Normal distribution

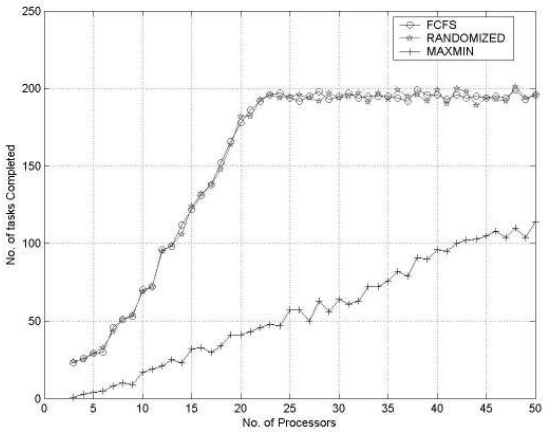

Figure 4. Comparisons with periodic tasks.

5. Conclusions and Future Research

This paper finds the number of optimal number of processor to be selected for a particular set of task to achieve maximum speedup. The results of our finding can be used to design an adaptive dynamic scheduler that selects the best strategy depending on load at a particular time frame. The results are also useful in deciding the effective group size of a processor pool (cluster) for the HDCS, which can be remodeled as a tree of resource clusters that are geographically distributed. If HDCSs are properly designed and planned, they can provide a more economical and reliable approach than that of centralized processing systems. There are many different types of distributed computing systems and many challenges to overcome in successfully designing one. The main goal of a distributed computing system is to connect users and resources in a transparent, open, and scalable way. Ideally this arrangement is drastically more fault tolerant and more powerful than many combinations of stand-alone computer systems. We are working on different falut tolorent scheduling schemes on HDCS that can yield optimnal performance in presence of different faults.

6. Acknowledgement

This research was supported by R&D project grant 2005 of MHRD Government of India with the title as "*Fault Tolerant Real Time Dynamic Scheduling Algorithm For*

Heterogeneous Distributed System" and being carried out at department of Computer Science and Engineering, NIT Rourkela.

7. References:

[1] G. Attiya and Y. Hamam, "Two Phase Algorithm for Load Balancing in Heterogeneous Distributed Systems," *Proceedings of the 12th IEEE Conference on Parallel and Distributed and Network Based Processing*, pp.18–27, March 2004.

[2] M. S. Iyengar and M. Singhal, "Effect of Network Latency on Load Sharing in Distributed Systems", *Journal of parallel and distributed Computing*, vol. 66, no. 6, pp. 839–853, June 2006.

[3] K. Y. Kabalan, W. W. Smari, and J. Y. Hakimian, "Adaptive Load Sharing in Heterogeneous Systems Policies, Modifications, and Simulation", *International Journal of SIMULATION*, vol. 3, no.1-2, pp.89–100, September 2005.

[4] Y. A. Li and J. K. Antonio, "Estimating the Execution Time Distribution for A Task Graph in a Heterogeneous Computing System," *IEEE Proceedings of Heterogeneous Computing Workshop*, pp. 335–346, August 1997.

[5] A. J. Page and T. J. Naughton, "Framework for Task Scheduling in Heterogeneous Distributed Computing Using Genetic Algorithms," *Artificial Intelligence Review*, vol. 24, pp. 415–429, November 2005.

[6] K. Savvas and M.-Tahar Kechadi, "Dynamic Task Scheduling in Computing Cluster Environments," *Proceedings of the ISPDC/HeteroPar IEEE conference*, pp. 121–154, March 2004.

[7] D-Tzen Peng, Kang G. Shin, and Tarek F. Abdelzaher, "Assignment and Scheduling Communicating Periodic Tasks in Distributed Real Time Systems," *IEEE trans. On computers*, vol. 23, no. 12, pp-745-757, December 1997.

[8] N. D. Thai, "Real time scheduling in distributed system," *International Conference on Parallel Computing in Electrical Engineering*, pp. 165–170, September 2002.

[9] Y. Zhang, K. Hakozaki, and K. Shimizu, "A Performance Comparison of Adaptive and Static Load Balancing in Heterogeneous Distributed Systems," *IEEE International Conference*, pp. 241–242, July 1995.

[10] Gerard Tel, "*Introduction to Distributed Algorithms",* Cambridge University Press, 1994.

[11] Tracy D. Braun, et' al., "*A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems",* Journal of Parallel and Distributed Computing, vol. 61, pp.810-837, 2001.

[12] Xiaoshan He, Xian-He Sun, and Gregor von Laszewski, "*QoS Guided Min-Min Heuristic for Grid Task Scheduling*", Journal of Computer Science and Technology, Special Issue on Grid Computing, vol. 18, no. 4, pp. 442 – 451, 2003.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, 1979.

[14] A. Ghafoor and J. Yang, "*Distributed heterogeneous supercomputing management system"*, IEEE Computer. Vol.26, no.6, pp.78-86, June 1993.

[15] M. Kafil and I. Ahmad, "*Optimal task assignment in heterogeneous distributed computing systems"*,IEEE Concurrency, Vol. 6, no.3, pp.42-51, July-Sep 1998.

[16] M. Pinedo, ``*Scheduling: Theory, Algorithms, and Systems",* Prentice Hall, Englewood Cliffs, NJ, 1995.

[17] H. Singh and A. Youssef, "*Mapping and scheduling heterogeneous task graphs using genetic algorithms*", in ``5th IEEE Heterogeneous Computing Workshop (HCW '96)," pp. 86_97, 1996.

[18] Muthucumaru Maheswaran et al, "*Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems*", Journal of Parallel and Distributed Computing vol. 59, pp.107-131,1999