50th Technical Review of Institution of Engineers, Orissa State Centre, pp. 113-118, 2009

Modeling and Analysis to Estimate the Performance of Heterogeneous Cluster

Pushpendra Kumar Chandra , Bibhudatta Sahoo, Sanjay Kumar Jena

Department of CSE , National Institute of Technology Orissa, INDIA

Abstract. The goal of load balancing is to assign to each node a number of tasks proportional to its performance. Many load balancers have been proposed that deal with applications with homogeneous tasks; but, applications with heterogeneous tasks have proven to be far more complex to handle. Load balancing techniques play a very important role in developing high-performance cluster computing platforms. Many load balancing polices achieve high system performance by increasing the utilization of CPU, memory, or a combination of CPU and memory. However, these load-balancing policies are less effective when the workload comprises of a large number of I/O-intensive tasks and I/O resources exhibit imbalanced load. The I/O intensive tasks running on a heterogeneous cluster needs effective usage of global I/O resources. We have proposed a load-balancing scheme based upon system heterogeneity and migrate I/O-intensive tasks to the fastest processor. The proposed load balancing scheme can minimizes the average slow down of all parallel jobs running on a cluster and reduces the average response time of the jobs**.**

Keywords: Heterogeneous cluster, I/O-intensive task, Load balancing

1 Introduction

Load balancing (LB) is a critical issue in parallel and distributed systems for the efficient utilization of the computational resources. There is a large body of literature on load balancing and all the proposed load balancing algorithms can be broadly characterized as static and dynamic. The focus of this paper is on the dynamic load balancing algorithms and the processing times of the jobs are known at the time of execution. Load balancing can be static or dynamic.

In static scheduling, the assignment of the tasks to the nodes is done before the execution of the program. Information regarding task execution time and processing resources is assumed to be known at compile time. A task is always executed on the node to which it is assigned. Dynamic scheduling is based on the re-distribution of processes among the processors during execution time. This redistribution is performed by transferring tasks from heavily-loaded processors to lightly-loaded processors with an aim to minimize the processing time of the application. The advantage of dynamic load balancing over static scheduling is that the system need not be aware of run-time behavior of the application before execution. The flexibility inherent in dynamic load balancing allows for adaptation to unforeseen application requirements at run-time. The major disadvantage of dynamic load balancing schemes is the run-time overhead due to:

[1] The load information transfer among processors,
[2] The decision-making process for the selection of processes and processor for job transfers, and
[3] The communication delay due to task relocation itself.

Dynamic LB algorithms can be further classified into a centralized approach and a decentralized approach. In the centralized approach only one node in the distributed system acts as the central controller. It has a global view of the load information in the system, and decides how to allocate jobs to each of the nodes. The rest of the nodes act as slaves; they only execute the jobs assigned by the controller. The centralized approach is more beneficial when the communication cost is less significant, e.g. in the shared-memory multi-processor environment.

The main motivation of our study is to propose a centralized dynamic LB algorithm that can cater for the following unique characteristics of practical distributed Computing environment:

- Heterogeneous system: There may be a difference in the hardware architecture, operating systems, computing power and resource capacity among sites. In this study, heterogeneity only refers to the processing power of site.
- Effects from considerable communication delay: The communication overhead involved in capturing load information before making a dispatching decision can be a major issue negating the advantages of job migration. We should not ignore the considerable dynamic communication delay in disseminating load updates.

Most load balancers were designed to handle applications with homogeneous tasks, for example data parallel application or tree-based algorithms. A lot of applications however consist of heterogeneous tasks, i.e. tasks performing different operation or operating on different types of data. Due to uneven job arrival patterns and unequal computing capacities and capabilities, the computers in one node may be overloaded while others in a different node may be under-utilized. It is therefore desirable to dispatch jobs to idle or lightly loaded computers to achieve better resource utilization and reduce the average job response time.
The rest of the paper is organized as follows. In the section 2 that follows, related work in the literature is briefly reviewed. In section 3, we describe the system model. In section 4 we describe the novel load balancing algorithm. Finally concludes the paper by summarizing the main contribution of this paper.

2 Related Work

In the past decade, load balancing techniques in the context of CPU and memory resources has been extensively studied in recent year. There are many approaches to balancing load in disk I/O resource can be found in literature [1][2][3][4][6][10]. Xiao Qin[1] proposed an algorithm IOLB and compares this algorithm with conventional CPU- and memory-aware load balancing schemes and shows that the IOLB algorithm significantly improves the resource utilization of a cluster under I/O-intensive workload. Mais Nijim Tao Xie, 2005 developed a performance model for self-manage computer systems under dynamic workload condition, where both CPU- and I/O-intensive applications are running in computer systems. They show that the controller is capable of achieving high performance for computer systems under workloads exhibiting high variability. Xiao Qin et al.[4] proposed a feedback control mechanism to improve the performance of a cluster by adaptively manipulating the I/O buffer sizes. The primary objective of this mechanism is to minimize the number of page faults for memory-intensive jobs while improving the buffer utilization of I/O-intensive jobs. The feedback controller judiciously configures the weights to achieve an optimal performance. Meanwhile under a workload where the memory demand is high, the buffer sizes are decreased to allocate more memory for memory-intensive jobs, thereby leading to a low page-fault rate. Increasing attention has been drawn toward I/O-intensive application. Kandaswamy et al. [10] examined optimization techniques and architecture scalability. They evaluated the effect of the techniques using five I/O-intensive applications from both small and large applications domain. Xiao Qin et al.[6] developed two effective I/O-aware load-balancing schemes, which make it possible to balance I/O load by assigning I/O-intensive sequential and parallel jobs to nodes with light I/O loads. However, the above techniques are insufficient for automatic computing platforms due to the lack of adaptability. We proposed an algorithm that take all the parallel task and it balance the I/O-intensive load with effective manner.

3 System Model

In our study we have considered a cluster computing platform of heterogeneous system in which set of N= {N1 ,N2 , N3 …..Nn} n nodes are connected via a high speed network. Each node in this model composed of a combination of various resources including processor, memory, disk ,network connectivity and every node is differ with their processor, memory and disk. A load manger or master node is responsible for load balancing and monitoring available resources of the node. Figure 1 shows the queuing model for load manager.

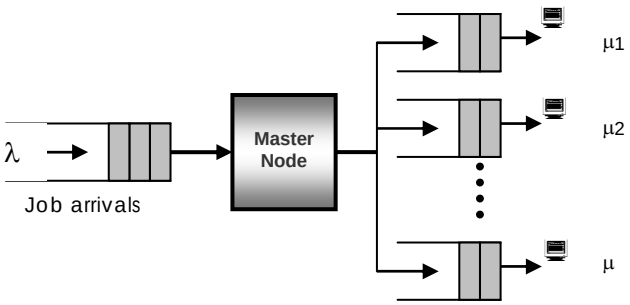

Figure 1: M/M/n heterogeneous system

Here we are considering a variant M/M/n queue where the service rates of the two processors are not identical this is the case of heterogeneous multiprocessor system. The queuing structure is shown in below figure. Assume without loss of generality that μ1> μ2 > μ3> μ4……..> μn

The state of the system is defined to be the tuple (k1,k2,k3……kn) where n1≥0 denotes the number of jobs in the queue including any at the faster processor and n2 denotes the number of jobs at slower processor. Jobs wait in line in the order of their arrival. When a processor is ideal, the faster processor is scheduled for service before the slower processor.

The traffic intensity for this system is

$$\rho = \frac{\lambda}{\sum_{i=1}^{n} \mu_i}$$

The average number of jobs in the system may now computed by observing that the number of jobs in the system. Therefore the average number of jobs is given by:

$$E[N] = \frac{1}{A(1-\rho)^2}$$

Where

$$A = \frac{(1+2\rho)\prod_{i=1}^{n} \mu_i}{\lambda \sum_{i=1}^{n} (\lambda + \mu_i)} + \frac{1}{1-\rho}$$

The prediction scheme consists of two parts. In the first part, which is an off-line procedure, resource usage states are determined for program executions of a given UNIX system. Resource usage data is collected for all processes that ran on the system for a few days, this data is analyzed as follows: Each process is represented by a point in a three-dimensional space, where each dimension corresponds to the resources of the system, i.e., the CPU, the memory, and the file I/O. A statistical clustering algorithm is then used to identify the high density regions of this three-dimensional space (i.e., determine the number of such regions and the means of their centroids). By definition, most program executions occur in or near these regions, and therefore they are referred to as the resource usage states.

In the second part, which is an on-line procedure, actual prediction is made. The prediction scheme builds and maintains a state-transition model for each program on an on-going basis. The states of the model are the resource usage states defined above. Suppose a program has been executed several times, providing a sequence of execution instances. First, the sequence of execution instances is converted into a sequence of resource usage states by assigning the nearest resource usage state to each execution instance. The state transition probabilities are then calculated from this new sequence to build a state-transition model for the program. The prediction is a weighted

mean calculation of resource requirements using the program's current state-transition model and the actual resource usage in its most recent execution. See [7] for further details. Then predicted value is fed to the selector that is used to select the best node among all nodes where the task will execute. That node is under-loaded and gives response effectively. Scheduler is responsible to dispatch the task to the node selected by the selector. Then task will send to that node and task will execute there. Load manager update the load status table.

4 Proposed Algorithm

We proposed an algorithm for a wide variety of workload conditions including I/O-intensive, CPU-intensive and memory-intensive load. The objective of the proposed algorithm is to balance the load of three types of resources across all nodes in a cluster. In this study analytically evaluate the performance of algorithm; we are focused on a remote execution mechanism in which task can be running on a remote node where it started execution. Thus preemptive migrations of tasks are not supported in our algorithm.

To describe this algorithm first we introduce the following three load indices with respect to I/O, CPU, memory resources. (1) CPU load of a node is characterized by the length of CPU waiting queue, denoted as LCPU(i). to identify whether node i's CPU is overloaded. (2) Memory load of a node is the sum of the memory space allocated to all the tasks running on that node. The memory load of node i is denoted as LMEM(i) (3)I/O load measures two types of I/O accesses, i.e. (a) implicit I/O request includes by page fault; (b) explicit I/O request issued from tasks. IO load index of node i is denoted as LIO(i). Table 1 shows the definition of notation we used in this paper.

Table 1: Definition of Notation

Notation	Definition
N	Number of node in heterogeneous system
j	Task submitted to the system
λ	Arrival rate of task
μn	Service rate of heterogeneous system
$IOREQ_j$	I/O requirement of task j
$CPUREQ_j$	CPU requirement of task j
$MEMREQ_j$	MEMORY requirement of task j
L_a^{IO}	I/O load on node($1 \le a \le n$)
L_a^{CPU}	CPU load on node($1 \le a \le n$)
L_a^{MEM}	MEMORY load on node($1 \le a \le n$)
L_{IO}^k	I/O load index on set of k node that satisfy all requirements
L_{CPU}^k	CPU load index on set of k node
L_{MEM}^k	MEMORY load index on set of k node

R_j^k	Response time of task on set of k nodes

Now we describe the load balancing algorithm of which the pseudo code is given above. Given a set of independent tasks submitted to the load manager. Our algorithm make an effort to balance the load of the cluster resource's by allocating each task to a node such that the expected response time is minimized. For each task j, our algorithm repeatedly performs steps 2-19 described follows:

First it will predict all three IOREQj, CPUREQj, MEMREQj requirements of task j from set of task by step 2. This three predicted value are important because according to this value task execute with best suited node. Step 3 is used to find the highest requirements of task and it is responsible for initiating the process of balancing I/O resources. Steps 4-7 are used to balance the I/O load. In step 4, if the I/O requirements of task j are high then it will find the set of nodes where I/O load is minimum and satisfies all the three requirements of the task. Step 5 calculates the response time of task with all selected nodes. In Step 6, if the response time is minimum with particular node then task will be sent to that specific node.

Algorithm: Load balancing

Input: a job with task *j* submitted to master node

1. for each task do
2. Predict the value of IO,CPU and memory requirements
3. if $IOREQ_j = \max(IOREQ_j, CPUREQ_j, MEMREQ_j)$
4. choose set of k node such that node $L_{IO}^k = \min_{a=1}^{n}(L_a^{IO})$ satisfy the all three requirements
5. calculate response time R_j^k of task j in set of k node
6. if $R_j^i = \min_{b=1}^{k}(R_j^b)$ then
7. dispatch the task to node N_i and execute there
8. else if $MEMREQ_j = \max(IOREQ_j, CPUREQ_j, MEMREQ_j)$
9. choose set of k node such that node $L_{MEM}^k = \min_{a=1}^{n}(L_a^{MEM})$ satisfy the requirements
10. calculate response time R_j^k of task j in set of k node
11. if $R_j^i = \min_{b=1}^{k}(R_j^b)$ then
12. dispatch the task to node N_i and execute there
13. else if $CPUREQ_j = \max(IOREQ_j, CPUREQ_j, MEMREQ_j)$
14. choose set of k node such that node $L_{CPU}^k = \min_{a=1}^{n}(L_a^{CPU})$ satisfy the requirements
15. calculate response time R_j^k of task j in set of k node
16. if $R_j^i = \min_{b=1}^{k}(R_j^b)$ then
17. dispatch the task to node N_i and execute there
18. update the load status;
19. end for

Second, in step 8, if the memory requirements of task are high then it will perform steps 9-12 to balance memory load among all the nodes. Page fault behaviors occur when the memory space allocated by running tasks exceeds the amount of available memory. That's why, it is necessary to balance memory to minimize the page fault. Step 9 searches the set of nodes with minimum memory load and satisfies all the three resource requirements of the task.

Step 10 calculates the response time of the task with all selected node. Step 11 finds the minimum response time of the task from selected node. Step 12 dispatches the task to selected node.

Third, step 13 is responsible if the CPU requirement of the task is high and step 14 searches the set of nodes with minimum CPU load among all the nodes that satisfy all requirements of the task. And then calculate the response time of the task in each selected node. Step16 finds node that gives minimum response time to execute the task. Step 17 dispatches the task to the selected node. Lastly, step 21 maintains updated load information that is send to the load manger.

5 Conclusion

There are number of different dynamic load balancing techniques for cluster systems; their efficiency depends on topology of the communication networks that connects nodes. This research has developed an efficient load balancing for I/O-, CPU- and MEMORY-intensive tasks. For this we developed a new way to predict and calculate the load of cluster nodes. The proposed load balancing scheme aim to achieve the effective usage of global disk resources in cluster. This can minimizes the average slow down of all parallel jobs running on a cluster and reduce the average response time of the jobs.

Future studies can be performed in following direction. First, we will evaluate the performance of scheme on a large scale of cluster. Second, we have assumed the task is independent, so we will also simulate this scheme for inter-dependent task. Third, in this study we have assumed network communication cost is negligible; therefore we will extend this to balance the load in network resource.

Acknowledgments. This research was supported by R&D project grant 2005-2008 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

References

1. Xiao Qin, Performance comparisons of load balancing algorithms for IO-intensive workloads on clusters, Journal of Network and computer applications(2006), doi:10.1016/j.jnca.2006.07.001
2. Xiao Qin ,Dynamic Load Balancing for IO-Intensive Tasks on Heterogeneous Clusters, Proceeding of the 2003 International Conference on High Performance Computing(HiPCO3)
3. Xiao Qin ,Hong Jiang ,Yifeng Zhu ,David R. Swanson ,A Dynamic Load Balancing Scheme for IO-Intensive Applications in Distributed Systems, Proceeding of 2003 international conference on Parallel processing Workshop(ICPP 2003 Workshop)
4. Xiao Qin, A feedback control mechanism for balancing I/O-intensive and memory-intensive applications on cluster, parallel and distributed computing practices journal
5. Paul Werstein ,Hailing Situ and Zhiyi Huang , Load balancing in cluster computer, Proceeding of the seventh international conference on Parallel and Distributed Computing, Applications and Technology (PDCAT'06
6. Xiao Qin, H.Jiang, Y.Zhu and D.swanson, Toward load balancing support for I/O intensive parallel jobs in a cluster of workstation, Poc. Of the 5th IEEE international conference cluster computing(cluster 2003) ,Hong Kong, Dec. 1-4-2003
7. Kumar K. Goswami, Murthy Devarakonda and Ravishankar K. Iyer, Prediction–baesd dynamic load-sharing heuristics, IEEE transaction on parallel and distributed systems, VOL.4, No.6, june 1993
8. Xiao Qin, An availability-aware task scheduling strategy for heterogeneous systems, IEEE transaction on computers.
9. Mohammed Javeed Zaki, Wei Li, Srinivasan Parthasarathy, A Review of Customized Dynamic Load Balancing for a Network of Workstations

10. M. Kandaswamy, M.Kandemir, A.Choudhary, D.Benholdt, Performance implication of architectural and software techniques on I/O intensive application, Proc International conference parallel processing 1998
11. Neeraj Nehra, R.B.Patel, V.K. Bhat ,A Framework for Distributed Dynamic Load Balancing in Heterogeneous Cluster,Journal of computer science 3(1):14-24-2007
12. Marc H. Willebeek-LeMair , Strategies for Dynamic Load Balancing on highly parallel computer IEEE Transactions on parallel and distributed systems Vol. 4,No. 9, September 1993.
13. Bibhudatta Sahoo, S. Soma Sekhar, and Sanjay Kumar Jena, "Dynamic Load Balancing In Heterogeneous Distributed Systems Using Genetic Algorithm", Advances in Information and Communication Technology, Macmillan India Ltd., 2007, pp. 223-230.
14. K.Trivedi, Probability and Statistics with Reliability, Queuing and Computer Science Applications