ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE), Vol.2 , No.3, Pages : 18-21 (2013)
Special Issue of ICCSIE 2013 - Held during 24 May, 2013 Bangalore, India

An Improved Adaptive Hierarchical Scheduling Policy for Processor Allocation in Heterogeneous Multicluster Systems

Gursimranjeet Kaur[1], Amit Chhabra[2], Harpreet Kaur[3]
[1]Deptt. Of Computer Science & Engineering, Guru Nanak Dev University, Amritsar, India, rabia.gem@gmail.com
[2] Deptt. Of Computer Science & Engineering, Guru Nanak Dev University, Amritsar, India, chhabra_amit78@yahoo.com
[3]Deptt. Of Computer Science & Engineering, Guru Nanak Dev University, Amritsar, India, harpreetaneja23@gmail.com

Abstract : Space sharing and time sharing have been traditionally used for scheduling in multiprocessor systems. At medium to heavy loads, time sharing policies perform well. However, at these loads, space sharing policies lead to wastage of resources and premature queuing of jobs, but perform well at low loads. In this paper, we investigate the previous work done in hybrid approaches to schedule in multiprocessor systems which is called hierarchical scheduling. Hierarchical scheduling policies eliminates the short comings of space sharing and time sharing while retaining the advantage of both the policies. We also propose an adaptive hierarchical scheduling policy for scheduling parallel jobs in heterogeneous multicluster systems.

Key words : Heirarchical Scheduling, multiprocessors, resource allocation, space and time sharing,.

INTRODUCTION

Cluster management packages help in load distribution among nodes in a cluster efficiently. Load distribution policies are classified into three types: static, adaptive and dynamic policies.

In static policy, processor once assigned, are held by a job until completion. This allocation is done at the time the job enters the system. The major advantage of the static policies is their ease of implementation and simplicity but its drawbacks are that it cannot consider changes in system state; therefore there is limited scope for improvement.

Adaptive policies on the other hand allocate the processors to the job depending on the system and workload conditions. Adaptive policy uses a combination of two policies; a receiver-initiated policy and a sender-initiated policy. At high loads, a receiver initiated policy might be used and at low to moderate system load, the load sharing policy is switched to sender-initiated.

The final category of dynamic policies, use the current system state information in making scheduling decisions. There is a lot of scope for performance improvements as compared to improvements obtained by the static policies, since these policies consider dynamic system changes. There are two types of dynamic policies: receiver initiated and sender initiated. In receiver initiated policies, the nodes which have low load search for the nodes with heavy load so that work can be transferred. Whereas in sender initiated policies, the heavily loaded nodes tries to transfer the workload to lightly loaded nodes.

Load sharing policy usually have two main components i.e. location policy and transfer policy. The work of transfer policy is to decide whether arrived job should be processed locally or at remote node. On the other hand the location policy determines the node to which job should be sent for remote execution. Most of the transfer policies make use of some load index threshold to check whether node is heavily loaded or lightly loaded. There are two kinds of location policies that can be used by load sharing policies. It can be either centralized policy or distributed policy.

In distributed policy, the load information of the system is distributed across the nodes in the system. In this policy, to locate a target node, a particular node would have to gather the load information from other nodes. Whereas in a centralized policy, a single node collects the state information and all other nodes consult this single node for determining a target node according to the system load. There are advantages and disadvantages associated with both policies. The major advantage of centralized policy is that it provides almost perfect load sharing. This is because the coordinator node has the overall system state information to make load distribution decisions. This policy lacks in fault tolerance and also has potential for performance bottleneck.

On the other hand distributed policy is fault tolerant and does not cause performance bottleneck as compared to centralized policy. But the main issue which can cause performance degradation is that whenever there is change in load or the system state information changes then it is transferred to all the nodes in the system. But this overhead can be decreased by sampling only a few randomly selected nodes. One more advantage of distributed policy is that it can be easily scalable to large system sizes.

To resolve above mentioned disadvantages, hierarchical scheduling policies were introduced which combines the merits of distributed and centralized policies while reducing the disadvantages of these policies. Moreover hierarchical scheduling policy eliminates the disadvantages of time sharing and space sharing policies while retaining their advantages.

In section 3, the work done in hierarchical load sharing policies has been investigated. In section 4 we proposed an improved adaptive hierarchical scheduling policy for scheduling jobs in inter as well as intra cluster systems. Section 5 concludes the paper.

LITERATURE SURVEY

In this section we will review the work done in hierarchical scheduling policies. Our choice of the research has been motivated by principles and techniques that can have strong influences on the performance of the distributed memory multicomputer systems.

Firstly we would like to define the appropriate terms in the following section so as to avoid any misinterpretation.

Relevant Definitions

- Adaptive Scheduling: In adaptive scheduling the decision of allocating job to a processor is made at scheduling time.
- Partition Reach: It is the number of processors which are reachable a cluster or node in the hierarchy.
- Partition Size: It is total number of processors in a given partition.
- Space Sharing: It is a scheduling technique in which processors are partitioned and jobs are scheduled on particular partitions till its completion.
- Time Sharing: It is a scheduling technique in which multiple jobs share a set of processors without any exclusion.
- System Utilization: It is the measure of percentage of time any processor in the system was busy.
- Response time: It is the time elapsed from the moment a job was submitted to the system till its completion.
- Execution Time: The time a job was being actually executed on a processor.

Previous Work

Sivarama P. Dandamudi and Philip **S.** P. Cheng in [1] proposed a task queue organization that combines the best features of centralized and distributed organizations. This task queue organization uses a hierarchy of queues, thus is named as hierarchical task queue organization. They also suggested that a carefully designed hierarchical organization leads to performance that is comparable to centralized organization. On the other hand it also eradicates the contention problem associated with ready queue. They also did analysis which determined and give guidance for designing hierarchical organization.

In [2] Thyagaraj Thanalapati and Sivarama Dandamudi proposed the Hierarchical Scheduling Policy (HSP) for scheduling in distributed memory multicomputer systems. They compared the HSP with a pure space- and time-sharing policy and observed the workloads at many high performance computing centers. Under many realistic considerations the detailed simulation results indicated a high performance of hierarchical scheduling policy over a range of workloads. There are various reasons for it:

1. Hierarchical Task Queue provides a way for removing contention from any one queue.

2. Hierarchical Task Queue, together with Hierarchical Scheduling Policy, gives a solution for getting over the problem of fragmentation typically originating with pure space-sharing policies.

3. Hierarchical Scheduling Policy allows for partial allocation of processors (that is it does time-sharing within an adaptively allocated partition)

The collective effect of all these characteristics is lead to observations of high utilization, lower response times and high degree of robustness.

In [3] Luyang Dong,Bin Gong, Yan Ma and Yi Hu proposed a hierarchical scheduling policy for large scale rendering. They implemented load balancing between resource dispatching and task selection to obtain desiring quality of services for rendering. They put forward a load balancing algorithm in which task execution and performance evaluation coincide depending upon dynamic feedback. The results present good improvement in terms of completion time as compared to non-strategy approach.

In [4] Sivarama P. Dandamudi and Thanalapati K. Thyagaraj have proved that the hierarchical scheduling policy outperforms the space sharing policy by a great margin. Also, it gives far better performance as compared to time sharing policy except at low system loads. As observed that for all practical cases, large parallel systems are unlikely to operate at low system loads, therefore the hierarchical scheduling policy provides significant performance improvements over the traditional policies.

In their implementation, the authors made an assumption that the processors are treated as a "pool of processors" in the system. For instance, the space sharing policy allocates four processors to the partition if there are four idle processors in the system and the partition size is four, no matter where these processors are residing in the system. If the system is completely bus-based, then only, this kind of allocation is fair. Large-scale distributed multicomputers are inclined to make use of hierarchical interconnection networks. In these kinds of systems, it is essential to assign nodes on a cluster-by-cluster basis. Their hierarchical scheduling policy executes processor allocations in this way. When these restrictions are applied on time sharing and space sharing, their performance outcomes will be much worse. Their hierarchical Scheduling policy has also been shown to give better performance in shared-memory NUMA systems.

In [5] Sivarama P. Dandamudi and **K.** C. Michael Lo have put forward a hierarchical load sharing policy that retains the advantages of centralized and distributed policies while minimizing the drawbacks associated with these policies. They have compared the performance of hierarchical load sharing policy to the distributed sender-initiated and receiver-initiated policies and centralized single coordinator policy. They also presented that the hierarchical load sharing policy yield significant performance improvements over the receiver-initiated and sender-initiated policies; it provides scalability and fault-tolerance near to that of a distributed policy while its performance is comparable to that of the centralized policy. They have not considered the impact of system and workload heterogeneity in their paper. System heterogeneity means that there are non-homogeneous nodes (nodes with different processing speeds) in the system and the workload heterogeneity means that job characteristics are non-homogeneous.

In [6] Sivarama P. Dandamudi and Michael Kwok Cheong Lo have put forward a new global hierarchical load sharing policy that reduces the drawbacks of the distributed and centralized policies while holding on to their advantages. They have taken into consideration a scenario where the bottleneck problem does not exist in the centralized policy, so that they can compare the performance of single coordinator policy with that of hierarchical policy. It has been confirmed by results that their proposed hierarchical load sharing policy provides better performance than the adaptive and distributed policies and shows performance very near to that of the centralized policy for various system and workload parameters taken into consideration in their study. They have also compared the performance of these policies in heterogeneous systems and the outcome of this proves that the hierarchical policy gives the best performance as compared to other policies mentioned above.

In [7] Michael Lo and Sivarama P. Dandamudi have done the comparison of the performance of two distributed policies and the centralized single coordinator policy with hierarchical load sharing policy. They have assumed that the scenario they are considering is the one with the centralized policy which does not have bottleneck problem, so that they can observe how close the single coordinator policy performs in comparison to the hierarchical policy. They have proved that the hierarchical policy provides good performance in the absence of contention i.e. it performs very similar to the single coordinator policy.

PROPOSED METHODOLGY

System Framework

In this section, we define the framework of our proposed scheduling policy. It is based on hierarchical organization. We assume that there are N number of nodes or workstations that can be clubbed to make clusters as shown in Fig 1. Workstations in clusters may differ in processing speeds (Basic Processing Units) i.e. they don't have same architecture. The Hierarchical Task Schedulers are organized logically in the form of cluster tree (tree of schedulers which is D-levels deep). The root node is the main scheduler (MS) and leaf nodes are the processors having their own local schedulers (LS). There are layers of schedulers in between main scheduler and leaf schedulers and these are called intermediate schedulers (IS). The number of children of any node is referred to as its branching factor B. We assume that branching factor is same for all the nodes in our system.

A node is in one of these states: sender, receiver or neutral state. A node is sender in case it has some task which is not assigned yet while a node is a receiver if it has initiated self scheduling. A node which is neither a sender nor a receiver is in neutral state.

Improved Adaptive Hierarchal Scheduling Policy

Our proposed policy is Improved Adaptive Hierarchical Scheduling Policy. In this policy the work will be transferred down the hierarchy on demand. We assume the system to be a shared heterogeneous system in which the workstations within a cluster (intra-cluster) also differ in Basic processing Units (BPUs). The closest approach to our work is that of J.H. Abawajy (2009). However there are several differences between his and our work:

- He is concerned with the shared heterogeneous cluster system in which nodes within a cluster are homogeneous, but we are considering heterogeneity within cluster too. As clusters are assumed to be scalable and while adding new nodes its convenient if heterogeneity is allowed within cluster.
- In addition we are considering breaking of job into a task depending upon a nodes branching factor, whereas he didn't consider branching factor in making any such decision. He just assumed that job will be broken down to task when it will reach last level of intermediate schedulers.
- Moreover we are considering scheduling jobs with CPU resource only, whereas he considered I/O resource scheduling too.

Proposed Approach

Jobs are submitted at the root node (Main Scheduler). MS queues the jobs in the wait queue until they are transferred to lower level schedulers on demand. When a request for task is received by the root node, then it transfers some of its jobs down to a particular scheduler depending upon the branching factor. If number of jobs present at the root node are greater than or equally to the branching factor then jobs are transferred without breaking into tasks otherwise task transfer takes place.

When a processor is idle i.e. its local scheduler does not have any task then it arise a request for task to its parent scheduler (IS). In case its parent also lacks in task it further requests its parent for a positive number of task transfer. This is followed recursively until request is satisfied or request reaches main scheduler.

Now when jobs/tasks reaches lowest level scheduler which will be last level intermediate cluster then tasks cannot be allotted equally because of different processing speeds of workstations within a cluster. Thus task allocation decision will be made according to the BPUs of a node. Nodes having higher number of BPUs will get more tasks than as compared to nodes with lesser number of BPUs.
So the job (say J) is broken down into task and allotted as:

Task transferred to P_1 = [(BPU of P_1) ⁄ (Total No. of BPUs within cluster)] * J

Algorithm

We have a queue for storing unscheduled tasks (i.e. QUEUE (job)) and a queue for storing pending requests for task transfer (QUEUE (RTT). If level (root) returns true then it means node is the main scheduler and in case level(leaf) returns true then node is a local scheduler.

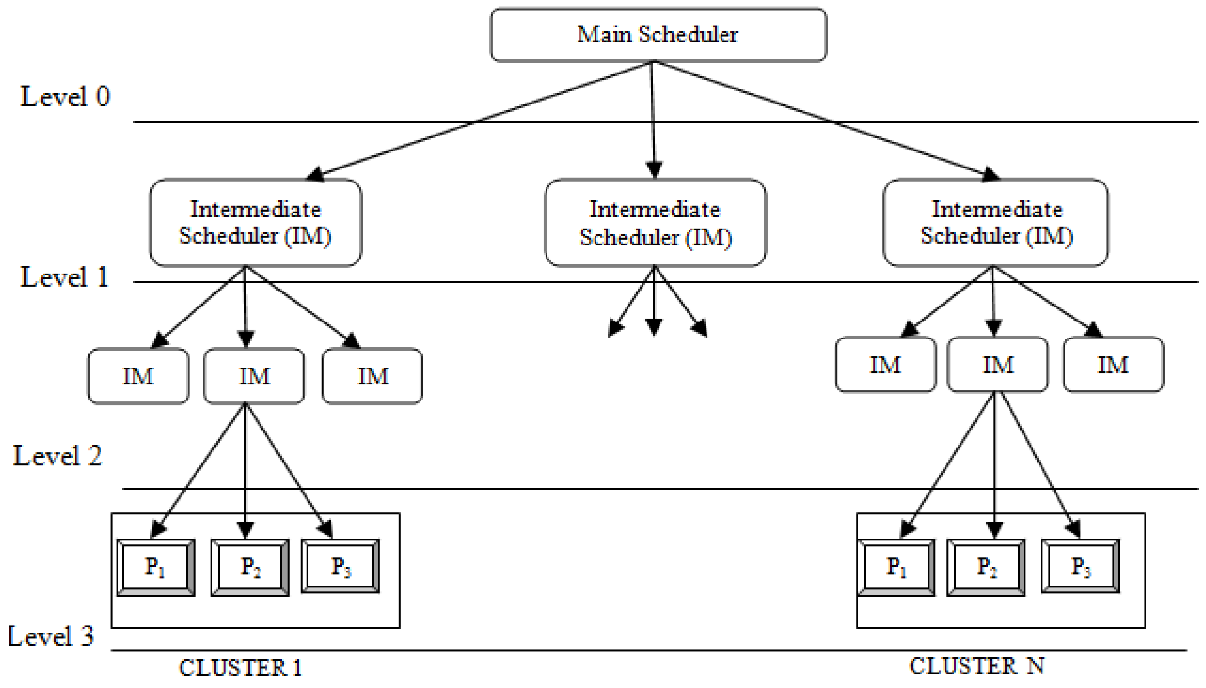

Fig 1. An overview of the Hiearchical Scheduling System architecture

Algorithm 1. Improved Adaptive Hierarchical Scheduling.

```
1: if state(neutral)==true then
2:    if level(root)==true then
3:       if (QUEUE(job)== null) then
4:          Put the request in wait queue
5:       else
6:          if No of jobs >= BF then
7:             Perform job transfer
8:          else
9:             Perform task transfer
10:         end if
11:      end if (level(root)==true) ∩ (level(leaf)==true) then
12:         if (QUEUE(job)== null) then
13:            if (QUEUE(RTT)== null) then
14:               Send job/task transfer request to parent node
15:            end if
16:            Put the request in wait queue
17:         else
18:            Perform Job/Task transfer depending upon BPUs
19:         end if
20:      else
21:         Send job/task transfer request to parent node
22:      end if
23:   end if
```

CONCLUSION

In this paper we discussed some techniques that are used in processor allocation to a parallel job. Then we pointed out the advantages and disadvantages associated with these technique thus declaring hierarchical scheduling policy (which combines the advantages of both space sharing and time sharing while eliminating their drawbacks), as a better policy than pure time shared and pure space shared policies. We also did the survey of the work done in hierarchical scheduling policies implemented in multiprocessor systems. So far, only inter cluster heterogeneity is considered in hierarchical scheduling policies. We extended this work by proposing an improved adaptive scheduling policy which takes into account inter as well as intra heterogeneity in clusters.

REFERENCES

[1] Sivarama P. Dandamudi and Philip **S.** P. Cheng, "A Hierarchical Task Queue Organization for Shared-Memory Multiprocessor Systems", *IEEE Transactions on parallel and distributed systems*, VOL. 6, NO. 1, JANUARY 1995.

[2] Thyagaraj Thanalapati and Sivarama Dandamudi, "An Efficient Adaptive Scheduling Scheme for Distributed Memory Multicomputers", *IEEE Transactions on parallel and distributed systems*, VOL. 12, ISSUE 7, JULY 2001.

[3] Luyang Dong,Bin Gong, Yan Ma and Yi Hu, "A Hierarchical Scheduling Policy for Large-scale Rendering", *Chinagrid Conference (ChinaGrid)*, 2011 Sixth Annual, 22-23 Aug. 2011.

[4] Sivarama P. Dandamudi and Thanalapati K. Thyagaraj, "A Hierarchical Processor Scheduling Policy for Distributed-Memory Multicomputer Systems", in *Proc. Fourth International Conference on high performance computing*, 18-21 Dec 1997.

[5] Sivarama P. Dandamudi and **K.** C. Michael Lo, "A hierarchical Load sharing policy for distributed system", in *Proc. of the 5th international Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, MASCOTS. IEEE Computer Society,1997.

[6] Sivarama P. Dandamudi and Michael Kwok Cheong Lo, "A Comparative Study of Adaptive and Hierarchical Load Sharing Policies for Distributed Systems", in *Proc. Computers and Their Applications*, 1998, pp.136-141.

[7] Michael Lo and Sivarama P. Dandamudi, "Performance of Hierarchical Load Sharing in Heterogeneous Distributed Systems", in *Proc. Int. Conf. Parallel and Distributed Computing Systems*, Dijon, France, 1996.