Journal of Computer Science 7 (12): 1793-1797, 2011
ISSN 1549-3636

Balanced Scheduling of Independent File-Sharing Tasks in Heterogenous Environment

R.K. Ponsy, Sathia Bhama and S. Thamarai Selvi
Department of Computer Technology, Anna University, India

Abstract: Problem statement: To examine the strategies for scheduling of independent file-sharing tasks in a heterogeneous environment and the concept of load balancing. **Approach:** We propose hypergraph partitioning based strategy for the scheduling of non-critical jobs. This is done by scheduling the tasks that share tasks among them to the same processor. The tasks thus scheduled are employed to a load balancing scheme for balancing the load on the processors by considering the average load on all processors. **Results:** This strategy reduces the input output overheads among the tasks thus reducing the end-point contention. **Conclusion:** Thus the batch execution time on the processors is reduced.

Key words: Hypergraph partitioning, partitioning strategy, significant performance, output overheads among, approaches inherently, homogeneous platforms, scientific computing, satisfied assign, virtual organization, second stage

INTRODUCTION

The Grid is emerging as a wide-scale, distributed computing infrastructure that promises to support resource sharing and coordinated problem solving in dynamic, multi-institutional Virtual Organization. Grid scheduling involves three main phases: resource discovery, which generates a list of potential resources; information gathering about those resources and selection of a best set; and job execution, which includes file staging and cleanup.

In this study, we address the problem of scheduling the tasks in a heterogeneous environment. We propose a novel, hypergraph based approach along with load balancing. The main advantage of the hypergraph model is that a hypergraph can model asymmetric dependencies. The approach in this study formulates the sharing of files among tasks as a hypergraph and employs a strategy of two stages for scheduling of tasks and file transfers. In the first stage, tasks are scheduled to the processors using hypergraph partitioning method. In the second stage, load balancing is done so as to balance the load on the processors.

Related work: Earlier approaches inherently looked at homogeneous platforms. Our current work targets truly heterogeneous environments and uses efficient mapping of tasks onto heterogeneous compute clusters. Kaya and Aykanat (2006) have concurrently developed an iterative improvement based heuristic for scheduling tasks sharing files on heterogeneous systems (Kaya and Aykanat, 2006). Their work assumes a central master file server. They do scheduling of file sharing tasks in three phases by using hypergraph partitioning method.

Giersch *et al*. (2006; Fujimoto and Hagihara, 2004; Karypis and Kumar, 1998; Catalyurek *et al*., 2007) proposed several different heuristics which reduce the time complexity while preserving the quality of schedules. This scheduling decision is based on the greedy choices that depend on the momentary completion time of tasks. Iterative improvement heuristics have been widely used for scientific computing Vydyanathan *et al.* (2006) communities because of their effectiveness with good-quality results and efficiency with short runtimes.

MATERIALS AND METHOD

In this study we compare the performance of hypergraph algorithm with minmin algorithm for those tasks that share files between them.

Proposed architecture: The overall architecture of the load balanced scheduler is shown in Fig. 1.

Hypergraph partitioning: Here we describe the hypergraph partitioning strategy and we propose an algorithm for the task to processor mapping which reduces the replication of files across the processors. The steps involved in the partitioning and the calculation of execution time, the algorithm for mapping the tasks to processor and rescheduling of the tasks to balance the load among the processors are explained below.

Corresponding Author: Ponsy.R.K, Department of Computer Technology Anna University

Hypergraph partitioning strategy: The hypergraph partitioning strategy reduces the communication volume. It is influenced by the cost of transferring the files required for the particular task to execute, local bandwidth, remote bandwidth between the server and client, time taken to execute a program of size one byte (compute byte) and the degree of overlap between the tasks.

The hyper graph H = (V, N) is defined with V, the set of vertices and N the set of edges. The cost of transferring a file F, $Transfer_j$ for a task T is $Transfer_j$ (one_byte)= ($Prob_{first\_task}$/RemoteBW)+(1-$Prob_{first\_task}$)*:

(1-$Prob_{mapped\_to\_the\_same\_node}$)/ RemoteBW

Where:

RemoteBW = the I/O bandwidth between the storage node and the compute node

$Prob_{first\_task}$ = The probability that task T will be the first task to execute in the group

$Prob_{mapped\_to\_the\_same\_node}$ = The probability that T executes on a node where file F has already been transferred

Here we assume uniform probability distribution. Hence we have:

$Prob_{first\_task}= 1/s_j$

where, s_j=size of hyper-edge:

$Prob_{mapped\_to\_the\_same\_node}=1 / P$

where, $_P$ is the number of compute nodes.

With the assumption that computation time is linear with the file size, we calculate the estimated execution time as:

$TimeExecution_i = \sum_{fj€Fi} filesize(f_j) * (Transfer_j + 1/ LocalBW + Compute_{byte}) + Queue\_waiting\_time$

where, LocalBW is the I/O bandwidth between the local disk to the compute node and $Compute_{byte}$ is the compute cost of one byte. Here $TimeExecution_i$ is the estimated execution time.

So by assigning file sizes as hyper-edge costs, the proposed method reduces the communication cost.

An example of the batch of tasks and its hypergraph is illustrated in the Fig. 2.

The hypergraph partitioning method involves the mapping of tasks to the processor based on the file sharing between tasks. The communication volume is reduced in this process. If overlapping of files among the tasks are more than the tasks are mapped to the same processor.

Consider six tasks that have to be scheduled to two processors, P1 and P2. The file requirement of the tasks is shown in Fig. 2.a. Task1 is scheduled to P1. It requires files C, E and G. Now the Task3 is scheduled to the same processor P1 since two of its required files C and E are already available in P1. Task5 is also scheduled to P1 since all the required files A, E and G is already available in P1.

Task 2 and Task 4 are scheduled to P2 based on the policy above. Now we have to schedule Task6. Task6 requires the files A, B and D. Now the file A is available in P1 but files B and D are available in P2. Considering the degree of overlap and the transfer cost, Task6 is scheduled to P2.

Task to processor mapping algorithm:

Step1: Calculate the execution time of each of the task

Step 2: Order the task according to the execution time with the task having least execution time at the head of the queue

Step 3: Assign the first task to a processor so that the task suffers minimum, for example P1.

Step 4: Transfer the files needed for the task to execute

Step 5: Take the next task in the queue and assign it to the processor P1 if the files that are present in the processor P1 are also needed by the next task satisfying the condition, no. of files(that are yet to be transferred) needed by the task <= ceil (no. of files needed for the task execution/2)

Step 6: Transfer the other files needed for task execution

Step 7: If the above condition is not satisfied assign it to another processor such that the task will least suffer.

Step 8: Repeat the steps 5, 6 and 7 until all the tasks has been assigned to the processor.

Load balancing: The files are thus scheduled by the method of hyper graph partitioning, are checked for the constraint of load balancing. The processor's individual queue is checked along with the load on the processor. Consider the case of two processor p1, p2 with tasks t1-t4. In case of all the tasks requiring the same file would be scheduled by hyper graph partitioning to the same processor to reduce the communication time.

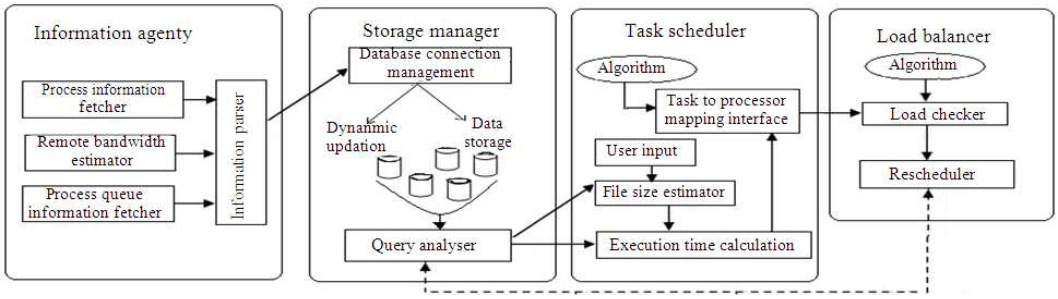


Fig. 1: Overall architecture of the system

But the queue waiting time would be large and load on one processor will be high. Hence the load balancer checks with the constraint and indicates rescheduler of the imbalanced load on the processor (Dhakal *et al*., 2007). Then two of the four tasks could be scheduled to other processor. Here we could finish the work faster. And comparing to the queue waiting time of tasks with the transfer time of files, we reschedule the tasks if it is reasonable.

Also if the processor has excess load, then we could transfer a group of tasks with similar file requirements to another processor. Each of the processor loads is compared with the average load of all other processors and balancing is maintained among all the processor's load (Dhakal *et al*., 2007).

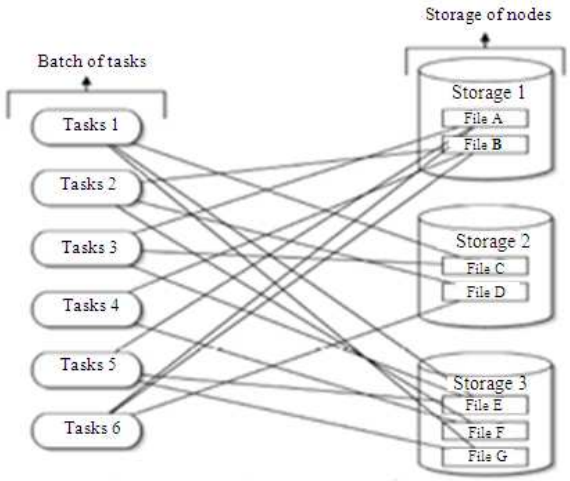
RESULTS AND DISCUSSION

We now present an experimental evaluation of the proposed hypergraph strategy along with MinMin algorithm. For experimental purpose we consider that all the files that are needed for the task execution is present in the server initially. The server has two client processors connected to it. The client submits the task to the server and the server schedules the tasks using hypergraph partitioning method so as to reduce the communication cost. The server contains the files A, B, C, D, E, F and G.

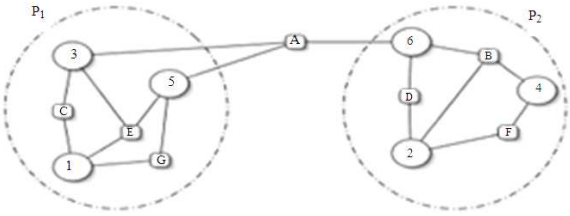
We have considered two cases here:

Case 1: The tasks submitted by the users share files between them. The number of files shared is more than half the number of files required for the task to execute.

Case 2: The tasks submitted by the users share files between them, but the number of files shared is less than the number of files required for the task to execute. For the case 1 we use the hypergraph partitioning method to reduce the communication between the compute nodes.



(a)



(b)

Fig. 2: Hypsergraph representatior of a sample batch of tasks. The numbers indicate tasks. The letters are files required by the tasks

For the case 2 we consider the memory available and load on the processor to schedule the task.

We now compare the degree of overlap with number of times a file is transferred from server to client. Figure 2 shows the result of comparison for the hypergraph partitioning method. Since in hypergraph partitioning the tasks that have high degree of overlap are assigned to the same processor, the number of times a file has to be replicated is minimized.

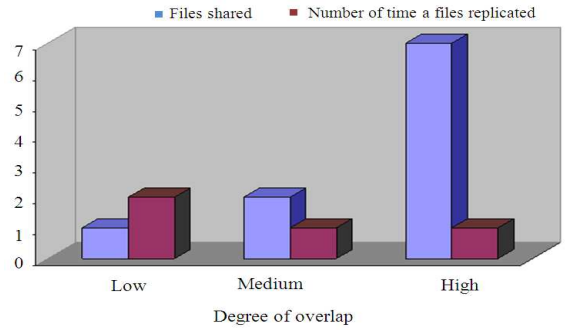


Fig. 3: Comparison of degree of overlap and number of times a file is transferred from server to client using hypergraph partitioning method

Table 1: The tasks share one file between them

Task name	Files needed
1	A, B, C
2	C, D, E
3	E, F, G

Table 2: The tasks share two files between them

Task name	Files needed
1	A, C, F
2	A, C, E
3	C, E, F

Table 3: The tasks share all the files that are required for the execution of each of the tasks

Task name	Files needed
1	A, B, C, D, E, F, G
2	A, B, C, D, E, F, G
3	A, B, C, D, E, F, G

In hypergraph partitioning, the file replication decreases with the increase in overlap. The number of times a file has to be replicated in hypergraph partitioning for lower degree of overlap is same as that in MinMin algorithm. To explain this, consider the values in Table 1. From this table it is clear that tasks 1, 2 and 3 share a file between them. In this case if we schedule all the three tasks to the same processor then for each of task we have to transfer two files table 2.

In order to improve the performance we can schedule those tasks whose degree of overlap is low by using the MinMin algorithm. This is because it would give a better throughput.

Figure 3 shows the result of comparison between the degree of overlap and number of times files is replicated for minmin algorithm.

For MinMin algorithm the number of times a file is replicated depends on the processor to which the tasks that share files between them are scheduled.

Consider the values in Table 3 the tasks share 7 files between them. By using MinMin algorithm, if tasks 1 and 2 are assigned to one processor and task 3 is assigned to the other processor, then all the seven files have to be replicated. Thus the number of times a file is replicated depends on the processor to which the tasks are scheduled.

CONCLUSION

The study developed a strategy for scheduling a collection of data intensive tasks using hypergraph partitioning method. Then load balancing is done to balance the load on the processors. The performance results shows that our strategy achieve significant performance over MinMin. Our future work would involve scheduling of attached jobs and integrating this scheduling with that of hypergraph partitioning method. Attached jobs are those which explicitly specify the processor where they want to run.

REFERENCES

Catalyurek, U.V., E.G. Boman, K.D. Devine, D. Bozdag and R. Heaphy *et al*., 2007. Hypergraph-based dynamic load balancing for adaptive scientific computations. Proceedings of the International Parallel Distributed Processing Symposium, Mar. 26-30, IEEE Xplore Press, Long Beach, CA., pp: 1-11. DOI: 10.1109/IPDPS.2007.370258

Dhakal, S., M.M. Hayat, J.E. Pezoa, C. Yang and D.A. Bader, 2007. Dynamic load balancing in distributed systems in the presence of delays: a regeneration-theory approach. Trans. Parallel Distribut., 18: 485-497. Pp: 10.1109/TPDS.2007.1009

Fujimoto, N. and K. Hagihara, 2004. A comparison among grid scheduling algorithms for independent coarse-grained tasks. Proceedings of the International Symposium on Application and Internet Workshop, Jan. 26-30, IEEE Xplore Press, Japan, pp: 674-680. DOI: 10.1109/SAINTW.2004.1268711

Giersch, A., Y. Robert and F. Viven, 2006. Scheduling tasks sharing files on heterogeneous master-slave platforms. 52: 88-104. DOI: doi:10.1016/j.sysarc.2004.10.008

Karypis, G. and V. Kumar, 1998. Multilevelk-way partitioning scheme for irregular graphs. J. Parallel Distributed Comput., 48: 96-129. DOI: 10.1006/jpdc.1997.1404

Kaya, K. and C. Aykanat, 2006. Iterative-improvement-based heuristics for adaptive scheduling of tasks sharing files on heterogeneous master-slave environments. IEEE Transact. Parallel Distributed Syst., 17: 883-896. DOI: 10.1109/TPDS.2006.105

Khanna, G., N. Vydyanathan, T. Kurc, U. Catalyurek and P. Wyckoff *et al*., 2005. A hypergraph partitioning based approach for scheduling of tasks with batch-shared I/O. Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid, (CCG'05), IEEE Computer Society Washington, DC., USA., pp: 792-799.

Vydyanathan, N., G. Khanna, U. Catalyurek, T. Kurc and P. Sadayappan, 2006. Scheduling of tasks with batch-shared I/O on heterogeneous systems. Proceedings of the 20th International Conference on Parallel and Distributed Processing, (PDP' 06), IEEE Computer Society, Washington, DC, USA., pp: 159-159.