TASK SCHEDULING IN GRID COMPUTING PLATFORM

Mrs. Nagamani T[1], Mr. Balamurugan V[2,] Mr.Senthil Kumar. V[3]
[1,2] *Assistant Professor, Department of Computer Science and Engineering, Bannari Amman Institute of Technology, Satyamangalam.*
[3]*Assistant Professor (Senior Grade) Bannari Amman Institute of Technology, Satyamangalam.*
[1]sennaga2005@gmail.com
[2]balamuruganvsrit@gmail.com
[3]civilvsk@gmail.com

***Abstract*—Grid computing is a high performance distributed computing environment to solve larger scale computational demands. It contains resource management, job scheduling, security problems, and information management and so on. Job scheduling is a fundamental issue in achieving high performance in grid computing systems.**

The objective of this project is to schedule independent, equal-sized tasks on a tree-based grid computing platform, where resources have different speeds of computation and communication. Instead of minimizing the total execution time, which has been proven to be Non-deterministic Polynomial time hard (NP-hard), we improve existing integral linear planning model. In this model, the time complexity to obtain optimal number of tasks assignment to each computing node of multi-level tree is high. To address this problem, Push-Pull method is proposed, which transforms the linear planning of multi-level tree into single-level tree and therefore the time complexity is greatly reduced. Based on the optimal tasks assignment to each node, a static distributed heuristic task scheduling algorithm is employed for establishing efficient mapping between tasks and available resources. The performance analysis of the resulting system reveals that the system is highly robust and efficient execution of the tasks is possible. The proposed approach employs a static distributed task scheduling algorithm for establishing efficient mapping between tasks and available resources. This scheduling strategy groups the user jobs according to a particular grid resource's processing capability and sends the grouped jobs to the resource. Job grouping in tree based grid environment enhances the computation/communication ratio.

***Keywords*—Grid Computing, Task Scheduling, Linear Planning, Optimal Scheduling Scheme, Distributed Scheduling Algorithm.**

I. INTRODUCTION

Grid computing systems are emerging as an important new field, different from conventional distributed computing systems by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation. The real specific problem that underlies the Grid concept is to coordinate the shared resources, and to solve problems through distributed programs. The sharing that the grid computing is concerned with is not primarily file exchange but rather direct access to computers, data, and other resources, as required by a range of collaborative problem-solving.

Task Scheduling deals with the master-slave paradigm on a heterogeneous platform, where resources have different speeds of computation and communication. More precisely, this deals the problem of allocating a large number of independent, equal-sized tasks in a tree based grid computing platform. Each node is a computing resource (a processor, or a cluster, or whatever) capable of computing and/or communicating with its neighbors at (possibly) different rates. We assume that one specific node, referred to as the master, initially holds a large collection of independent, identical tasks to be allocated on the grid.

The optimal scheduling scheme that determines the optimal number of tasks assigned to each computing node is obtained. This is done by pushing a multi-level tree up into a similar equivalent tree and then pulling this equivalent tree to obtain an optimal task assignment. A static distributed heuristic task scheduling algorithm is used. This approach reduces the total time taken in transmitting the user jobs to/from the resources and the overhead processing time of the jobs at the resources.

II. RELATED WORK

In this section, we briefly explain about some existing approaches. Our focus is to schedule the tasks in such a way that to minimize the completion time of all tasks. Subramani et al [2002] proposed a simple distributed duplication scheme for independent job scheduling in the Grid. A Grid scheduler distributes each job to the *K* least loaded sites. Each of these *K* sites schedules the job locally. When a job is able to start at any of the sites, the site informs the scheduler at the job-originating site, which in turn contacts the other *K*-1 sites to cancel the jobs from their respective queues. By placing each job in the queue at multiple sites, the expectations are improved system utilization and reduced average job makespan. The parameter *K* can be varied depending upon the scalability required.

Silva et al [2003] proposed a resource information free algorithm called Workqueue with Replication (WQR) for independent job scheduling in the Grid. The WQR algorithm uses task replication to cope with the heterogeneity of hosts and tasks, and also the dynamic variation of resource availability due to load generated by others users in the Grid. Unlike the scheme in [103] where there are no duplicated tasks actually running, in WQR, an idle resource will replicate tasks that are still running on other resources. Tasks are replicated until a predefined maximum number of replicas are reached. When a task replica finishes, other replicas are cancelled. In this approach, performance is increased in situations when tasks are assigned to slow/busy hosts because when a task is replicated, there is a greater chance that a replica is assigned to a fast/idle host. Another advantage of this scheme is that it increases the immunity to performance changing, since the possibility that all sites are changing is much smaller than one site.

Radulescu et al [2005] presented list heuristic algorithm called Fast Critical Path (FCP), intending to reduce the complexity of the list heuristics while maintaining the scheduling performance at the same time. The motivation of FCP is based on the following observation regarding the complexity of list heuristics. Basically, a list heuristic has the following procedures: the $O(e + v)$ time ranking phase, the $O(v \log v)$ time ordering phase, and finally the $O((e + v) \times p)$ time resource selecting phase, where e is the number of edges, v is the number of tasks and p is the number of resources. Usually the third term is larger than the second term. The FCP algorithm does not sort all the tasks at the beginning but maintains only a limited number of tasks sorted at any given time. Instead of considering all processors as possible targets for a given task, the choice is restricted to either the processor from which the last messages to the given task arrives or the processor which becomes idle the earliest. As a result, the time complexity is reduced to $O(v \log p + e)$.

Topcuoglu et al [2005] presented a heuristic called Heterogeneous Earliest-Finish-Time (*HEFT*) algorithm. The HEFT algorithm selects the task with the highest upward rank (an upward rank is defined as the maximum distance from the current node to the exiting node, including the computational cost and communication cost) at each step. The selected task is then assigned to the processor which minimizes its earliest finish time with an insertion-based approach which considers the possible insertion of a task in an earliest idle time slot between two already-scheduled tasks on the same resource. The time complexity of HEFT is $O(e \times p)$, where e is the number of edges and p is the number of resources. The experimental results show that the scheduling algorithms obtain better performance than other traditional algorithms.

III. PROBLEM DESCRIPTION

On tree-based grid computing platform, each computing node has zero or more son nodes, but has only one father node. We take the master-slave grid task scheduling model into consideration, following single-port model [3]. Most of network models can be simplified into tree-based models to resolve and make full use of parallel processing of grid computing platforms.

The task scheduling problem being discussed in this paper conforms to the following restriction: 1) tree-based heterogeneous grid computing platform; 2) considering the migration costs, that is to say, it takes time to transmit tasks; 3) each task is computed by one node; one computing node can run only one task each time; 4) using single-port master-slave scheduling model; 5) all the tasks waiting for scheduling are input into the root node. Figure.2 shows a single-level tree-based grid computing platform with 4 nodes, in which the weight of node indicates its computing capacity, whose value equals the number of time units that the node takes to run one task. The weight of edges indicates the communication capacity, whose value equals number of time units that it takes to transmit one task from one node to its son.

IV. SINGLE LEVEL TREE BASED SCHEDULING MODEL

A. Single Level Scheduling Model

Figure.1 shows general single level tree-based grid computing platform, consisting of k computing nodes. Units of time required for computing one unit task for each node are, respectively. Root node n_0 is Master, responsible for transmitting tasks to its sons. It takes $c_0, c_1, \ldots c_{k-1}$ units of time to transmit one unit task to its sons.

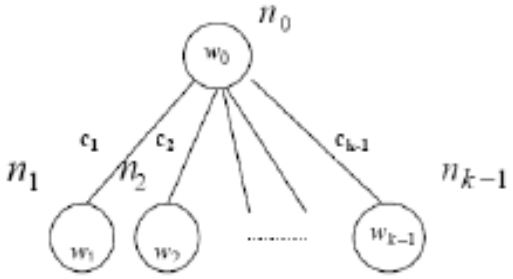

Figure 1. Two level grid computing platform

Paper [2] neglected restrictions that one task starts after completion of its communication from its father. The task assignment solution obtained using linear planning, is only the

upper bound of the optimal solution. To schedule M independent tasks of the same size on single level grid computing platform, the scheduling algorithm should know how many tasks should be assigned to the root node n_0, and how many should be sent to each son node separately, in order to minimize the execution time of all tasks, which is called the optimal solution of tasks assignment.

B. Task Assignment Example

Assume a single-level tree-based grid computing platform with 4 computing nodes, shown in Figure 2.

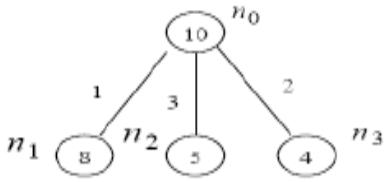

Figure2. Grid Computing Platform

To complete 8 tasks, model tasks assignment using the linear planning model, and obtain the optimal solution to this question, which is $\{x_0,x_1,x_2,x_3\}= \{1,2,2,3\}$.

Minimizing T satisfies the following conditions.

1) $x_0+ x_1+ x_2+x_3 = 8$
2) $10x_0 \leq T$
3) $8x_1 \leq T-1$
4) $5x_2 \leq T-3$
5) $4x_3 \leq T-2$
6) $x_1+ 3x_2+2x_{3+} 4 \leq T$
7) T, x_0, x_1, x_2, x_3 all are positive integers.

V. TASK ASSIGNMENT ON TREE BASED GRID

For multilevel tree, time complexity of tasks assignment algorithm to solve linear planning model is more. Costs to solve linear planning model will be very high. This paper introduces push-pull method. The push-pull algorithm first uses push operation to convert a two-level sub-tree into a equivalent node from leaf node to root node, which finally become a approximately equivalent tree with only one root node; then pull method is used to calculate the optimal number of tasks assignment to each new two-level sub-tree (called single level sub-tree), finally the approximately optimal solution of number of tasks assignment to each node in multi-level tree is obtained.

A. *Push Operation*

Push operation is defined as follows.

- Starting from leaf nodes with the same father node, a single-level tree consisting of a father node and its leaf nodes, is substituted by an equivalent node, whose computation and communication capacity is obtained by algorithm [4]. That is, first delete the leaf nodes of the single level tree in the original tree, then use an equivalent node to substitute the root node of the single level tree, which has the same computation and communication capacity, finally a new tree is created, and the equivalent node become a leaf node of T′
- Record sequence S of Push process and the link between equivalent node and its corresponding single level tree
- Repeatedly use step (1) and step (2) on the newly created tree in step (1), until a tree with only one node is left.

An example for push operation is shown below in Figure 3, Figure 4 and Figure 5. A multilevel tree with 6 nodes n_0 to n_6 is shown in Figure 3. The computation capacity of nodes n_0 to n_6 is w_0 to w_6 respectively and the communication time needed to transfer the unit tasks from n_0 to n_1 is c_0 and from n_0 to n_2 is c_1 and so on.

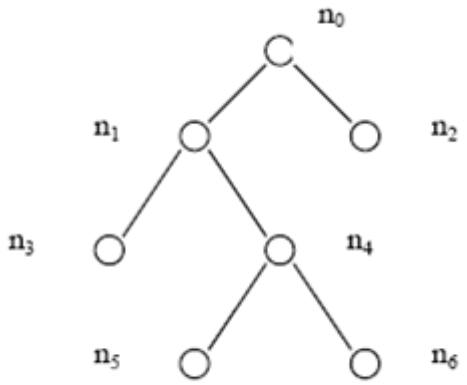

Figure 3. Multi-level tree

First push operation is shown in Figure 4 below. The single level tree consisting of n_4, n_5 and n_6 is converted into equivalent node n_4'. Computation capacity of n_4' is the computation capacity of the single level tree with nodes n_5 and n6 namely $w_4+w_5+w_6$.

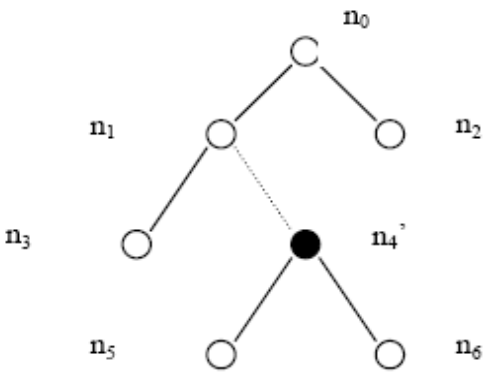

Figure 4. First Push

Second push operation is shown in Figure 5 below. The single level tree consisting of n_4', n3 and n_1 is converted into equivalent node n_1'. Computation ability of n_1' is equivalent to the computation capacity of n_4' and node n3 namely $w_1+w_4+w_5+w_6$

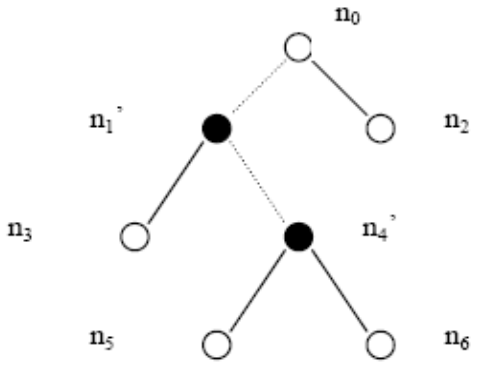

Figure 5. Second Push

B. Pull Operation

Pull is the inverse process of Push operation. First expand the tree of the final result of Push process inversely according to the sequence and finally the tree structure is recovered as before Push process starts.

We can obtain the approximately optimal tasks assignment by using push-pull method and simplified linear planning of multi-level tree into that of single level tree. To obtain the optimal number of tasks assigned to each node on tree-based grid platform by using linear programming model, we can use centralized algorithm which runs tasks on one node (usually root node), or distributed algorithm by using parallel computing ability in tree grid. Distributed algorithm for task assignment is given in the following.

```
task_assign (node)
Begin
        //If the node is root node of multi-level tree
    if (node is chief root node)
        //Equivalent tree algorithm, only run on chief
        //node n0;
    ETT ()
      // If the node is the root node of single level tree
    if (node is root node)
      foreach child
        // For all the son node
       // Send number of tasks nTask to son node i
      send(nTask,child[i]);
    else //the leaf of single level tree
       //Receive number of tasks nTask from root node
      receive(nTask,father);
End
```

C. Heuristic Static Task Scheduling Algorithm

As shown in Figure 6, each computing node has two time axes (computing and communicating). The one with an arrow indicates computing of tasks, the other without arrow indicates communication of tasks. In reference [2], the optimal solution

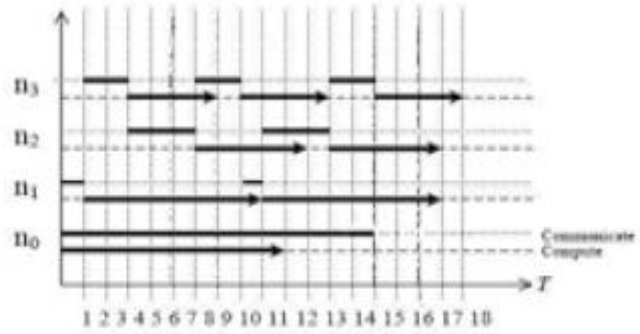

Figure 6. Optimal task Scheduling

obtained was 17. In fact, at t=13, was still communicating with, therefore, it couldn't start to run a task. This scheduling algorithm makes receive of task between son node and its father, and start of this task is sequential; both operations can not run in parallel. Since in this algorithm, only when a node becomes idle, it send request to its father node. We suggest a static heuristic algorithm, using the optimal tasks assignment as part of heuristic knowledge to dispatch tasks to son nodes, which removes the defects of OPCHAT and OPBHAT algorithm[2] mentioned above. This algorithm deals with single level tree. Multi-level tree can be converted to single level tree to use this algorithm

The task scheduling algorithm is divided into following steps:

- Candidate node set S is initialize to include root node and leaf nodes(all nodes are idle);

- When scheduling tasks, we pick up the most appropriate node from candidate set S, and assign task to, remove from candidate set S.
- The strategy of picking up an appropriate node from candidate set S is: first, consider whether this node gets optimal number of tasks assignment; according to the strategy of computation speed first OPCHAT(or communication bandwidth first OPBHAT), pick up one node from candidate set S, and assign a task to , remove from candidate set S.
- Update candidate set S, go to step (2), until candidate set S is empty and all task is assigned. Note that, when root node is idle, it is always assigned a task first.

Candidate set S should be updated when communication between root node and its son has completed. As Figure.6 shows, when t = 7, node has already got first task, at this time, root node updates task set S, schedule algorithm chooses next node waiting for a task.

VI. RESULTS AND DISCUSSIONS

GridSim has been used to create the simulation environment. The inputs to the simulations are total number of gridlets, average MI of Gridlets, MI deviation percentage, granularity size and Gridlet overhead time The MIPS of each resource is also specified (Table 1)

Resource	MIPS
R1	20
R2	44
R3	69
R4	296
R5	126
R6	210
R7	204

Table 1: MIPS of Grid Resources

This paper adopts GridSim tool to simulate the heuristic algorithm of task scheduling given above. GridSim provides a series of core function for the establishment and simulation of heterogeneous distributed computing environment, particularly suitable for simulation and research of task scheduling on grid. We simulated a treebased grid computing platform with seven nodes, five seconds of granularity time, five seconds of overhead time and 10% deviation. Then the platform used FCFS and tree-based scheduling algorithm with and without job grouping to schedule independent tasks of number of 50, 100, 150 and 200 of the same scale.

The Comparison results between FCFS and Tree based task scheduling algorithm of simulation is shown in Table2.

Number of Gridlets	FCFS (in seconds)	Task Scheduling (in seconds)
10	108	90
50	450	405
100	895	815
200	1670	1612

Table 2. Simulation time for different Gridlets

VII. CONCLUSION

This paper discusses job scheduling in heterogeneous tree-based grid computing environment. By doing research and analysis of this problem, that aims at task scheduling with minimum total tasks completion time on a multi-level tree grid computing platform, a new measure, called Push-Pull is used to build a single level tree, and develop a linear planning model for it. Through Push-Pull, the problems of optimal number of tasks assignment and task scheduling on a multi-level tree is iteratively converted to those of a groups of single level tree, which can be implemented in parallel on a tree grid platform.

GridSim is employed to carry out and simulate the tasks assignment algorithm, and distributed task scheduling. The results are compared with FCFS. The conclusion is that the scheduling algorithm employed is better than FCFS. This Static Heuristic Scheduling algorithm only takes the initial research on task scheduling in tree based platform. However many issues remain open. Further improvement should be done to handle more complicated scenario involving grouping of tasks and other QoS attributes. The improvement of this algorithm should concentrate on discussing simultaneous instead of independent task scheduling in heterogeneous tree-based grid computing environment.

REFERENCES

[1] Abraham A,Buyya R, Nath B. “Nature’s heuristics for scheduling jobs on computational grids”, Proc. Of the 8th International Conference on advanced Computing

and Communications (ADCOM 2000), New Delhi: Tata McGraw-Hill Publishing, 2000, pp.45-52.

[2] Ibarra OH, Kim CE. Heuristics algorithms for scheduling independent tasks on nonidentical processors. Journel of the ACM, 2006, pp.280-289.

[3] Braun TD, Siegel HJ and Beck N, "A comparison of eleven static Heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of parallel and Distributed Computing 2001, 61(6), pp.810-837.

[4] Casanova H.Simgrid: A toolkit for the simulation of application scheduling Proceedings of the 1st IEEE/ACM International symposium on Cluster Computing and the Grid. Brisbane: IEEE Computer Society Press, 2001, pp.430-437.

[5] Michael bender, "Flow and stretch Metrics for Scheduling Continuous Job Streams, Proceedings of the 9th Annual ACM-SINA Symposium on Discrete Algorithms, 2005.

[6] Dong F and Akl S, "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. Technical Report, http://www.cs.queensu.ca/TechReports/Reports/2006-504,2004.

[7] M.Maheswaran, S. Ali, H.J.Siegel, D. Hensgen and R. F. Freund, Dynamic Matching and Scheduling of a class System, Journel of Parallel and Distributed Computing, VOl.59, No. 2, pp.107-131, 2004.

[8] D.P.Silva, W. Cime and F.V. Brasileiro, "Trading Cycles for Information: using Replication to schedule Bag-of-Tasks Applications on Computing Grids, in Proceedings of Euro-Par 2003, pp.169-180, 2003.

[9] V.Subramani, R.Keeimuthu, S.Srinivasan and P.Sadayappan, Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests, in Proc. Of the 11th IEEE symposium on High Performance Distributed Computing (HPDC 2002), pp.359-366, 2002.

[10] H.Topcuoglu, S.Hariri and M.Y.Wu, Performance Effective and Low- Complexity Task Scheduling for Hetergeneous Grid Computing, IEEE Transactions on Parallel and Distributed Systems, Vol.13, No.3, pp.260-274, 2005.

