Grid Resource Allocation and Task Scheduling for Resource Intensive Applications

Abdul Aziz, Hesham El-Rewini
Computer Science and Engineering Department
Southern Methodist University, Dallas. TX
{aaziz, rewini}@engr.smu.edu

Abstract

Evolution of grid has drawn attention from various resource intensive applications addressing domains of bio-informatics, astrology and multimedia, to name a few. Image analysis applications in both bio-sciences and astrophysics are explicitly built in modular fashion to exploit the potential offered by grid. In this paper we present a framework for resource allocation and task scheduling, MARS, which efficiently allocates resources for such resource intensive applications which can be represented as Task Interaction Graph (TIG), while keeping makespan as minimum as the current state of resources in grid allows. Simulation experiments with MARS show a considerable improvement in job completion time and host utilization over traditional resource management and scheduling algorithms for grid.

1. Introduction

A grid is composed of a set of shared resources made available by different providers. Consumers can request grid management system for allocation of required resources according to their needs. Allocation of resources is a bi-party agreement as it depends both upon meeting consumer's needs and fulfilling provider's constraint. An example of provider set constraint is limit on extent of time a resource can be allocated.

Scheduling applications onto a grid is of special interest for resource intensive application. Normally such applications are modeled as directed acyclic graph (DAG). Representative of these applications come from various domains such as high-energy physics, bio-informatics, multimedia, and cryptography. Domains such as bio-informatics and astronomy rely heavily on image analysis for various purposes Some examples in this area are Grid-enabled Image Processing [1], EMAN [2], and Montage [3]. First two applications are closely related with biological systems whereas Montage deals with creating a wide view of area by combining images covering smaller areas, for use in astrology and astrophysics. Processing requirements of such applications are pretty intensive. Creating a mosaic image of Rho Ophiuchi dust cloud which consisted of 972 images, for example, takes 15 hrs on a 1GHz Sun machine.

Finding appropriate resources and completing the application in the minimum possible time is of prime interest for grid management and scheduling system. In this paper we present Management Architecture for Resource Services (MARS), which is a framework for resource allocation and task scheduling in grid environment. The rest of the paper is structured as follows. In Section 2 we present the problem statement. Section 3 serves as a comprehensive review of scheduling strategies and resource management in grid. In Section 4 we present MARS resource allocation and task scheduling mechanism. In Section 5 we present experimental evaluation of our proposed technique. Section 6 concludes the paper with pointing out the future directions.

2. Problem Formulation

A resource is an entity that can perform a useful function and maybe contributed to the grid. Providers are the owner of such resources and control access to them by defining access policy. Clients are consumers of the resources present in the grid. Resource(s) along with a communication interface is called a host or workstation. A host can have one or more resources. Every host has associated set of general and resource specific attributes and usage restrictions. The general attributes are used to identify host itself, whereas the resource specific attributes defines services provided by the host. A task is a unit of work which can be completed in isolation once all its resource requirements are completed. Every task has an associated set of required resources. Tasks also carry an estimated completion time if all the task requirements are fulfilled. A set of interacting tasks is called a job. A job can be represented as a directed acyclic graph called Task Interaction Graph (TIG). A Task Interaction Graph has tasks T as vertices and communication links C as edges. Formally a TIG is represented as (T, C). Directed edges between the tasks establish task precedence and the weight on these edges represent communication cost between the tasks. Communication between two tasks T_i and T_j is shown with the communication link C_{ij}. Such directed communication edge also indicates that T_j can not start until T_i finishes and send all the required communication to T_j.

The purpose of task scheduling algorithms is to map given number of tasks onto the available workstations, with respect to some objective function. Most common objective functions are makespan (C_{max}), total flow time (C_i), and weighted total flow time (w_iC_i). For the purpose of this paper we will deal mainly with the makespan of the schedule which is also known as schedule length. The allocation problem itself is shown to be NP-complete in general [4,5]. Allocation of precedence constraint task graph to multi processors is also proved to be NP-Complete [6,7,8], and NP-hard with additional constraints. The problem of optimally scheduling set of precedence constraint tasks on more than one workstation is shown to be NP-complete when all the tasks have unit time execution and unit time communication delays [9]. Various results proving NP-hardness of the problem with respect to identical and uniformly- related machines are reported by Garey and Johnson [10], Ullman [11] and Kubiak [12, 13].

Given a set of resources (R) and a job consisting of number of tasks (T), the main characteristics that define our task allocation algorithm are as follows:

- An online, preemptive task scheduling algorithm for precedence constraint tasks.
- For jobs arriving over time, with each job consisting of *i* number of *clairvoyant* tasks.
- Scheduling onto dynamic set of multi-purpose uniformly related distributed networked workstations (or more precisely for grid)
- While observing restricted assignment, i.e. only some of the available workstations fulfill the task requirements

The objective function of our proposed approach is to:

- Minimize the job completion time, makespan, C_{max}.
- Minimize the number of resources needed for the completion of job. For the purpose of this paper, we measure it as number of workstations allocated for a specific job.

There are two dimensions of this problem, first identifying the feasible resources for running the task, termed as matching, and second, ordering the execution of tasks on these resources, such that both the objective functions are met. Formally, there are n number of workstations (W), and m Jobs (J). A job can be composed of number of tasks where each task T can be processed by any workstation in the set μ_T such that $\mu_T \subseteq \{W_1, W_2, W_3, W_4, \ldots, W_n\}$. The composition and presence of workstations varies over time, given the dynamic environment of grid. Thus making discovery of resources and matching a crucial component of the overall approach. Another crucial restriction is on the processing time of the algorithm. An efficient scheduling algorithm for such environment should be fast in order to compete with any other algorithm which is working without any complex scheduling objective function, i.e. it will schedule the job on the first available matching workstation.

3. Related Work

Scheduling problem is known to be NP-complete in the general as well as many special cases. Various heuristics have been proposed for scheduling tasks onto the multi-purpose machines [14, 15, 16]. As our work spans two major areas of distributed computing, namely scheduling heuristics for heterogeneous resources and grid resource management, in the following sections we will cover the current state-of-art in both the areas.

3.1. Scheduling Heuristics for Heterogeneous Environment

Scheduling heuristics can be divided into four broad categories based on how they match tasks to workstations. Following is a brief review of the prominent classes in scheduling heuristics. Though genetic algorithms were able to produce a makespan which is better by 5 to 10%, their time consumption was much higher (with an average execution time of 60s) as compared to average running time in milliseconds for other static algorithms [14].

Experiments with scalability of genetic algorithms have found that performance of the algorithm declines when the problem size increases [17]. Another limiting factor for genetic algorithms is that a set of several control parameters need to be determined for producing acceptable schedule. A set of control parameters that works best for one problem may not work for the other. Clustering heuristics assume unbounded number of processors, and therefore final scheduled produced by such algorithms is subjected to further processing. Post processing for such algorithms generally includes reduction in the number of cluster to match with the processors available and ordering of tasks. Task Duplication based heuristics have higher algorithm complexity as compared to other heuristic classes. Unless required by the clients as quality of service parameter, scheduling tasks redundantly in grid may lessen the chances of scheduling any subsequent tasks.

Network of Workstations (or Grid) environment is characterized by its dynamic composition, heterogeneous resources, online arrival of jobs and restrictive task-to-host mapping. An ideal scheduling algorithm for grid should take into account these factors to create an acceptable schedule for incoming precedence oriented jobs. A common assumption which list scheduling heuristics make is static target environment for task mapping. Given the offline nature, where all the tasks are known in advance, list scheduling heuristics often resort to constructing a computation time matrix M, where M_{ij} gives the execution time of task t_i on processor p_j [18]. This approach can work well in a static environment with a small number of workstations but for environments such as grid, where the availability of resources is dynamic in nature and jobs arrive over time, constructing such a table is virtually impossible. List scheduling heuristics are generally less complex and generate better schedules than other categories

Another important factor which makes traditional scheduling algorithm un-usable in grid environment is the need of advance reservation mechanism. When a client is in need of set of resources to run a job, it advertises its requirements to multiple resource brokers. These resource brokers check if they can fulfill the resource requirements or not. If former is the case, then Resource Manager reserves these resources for the job and notifies the client. The client picks up the best offer it gets and releases the job to that resource manager. In this scenario it is not only necessary to reserve resources but to also provide mechanism for expiring reservations if resource manager does not receive the job.

3.2. Existing Grid Management Architectures

We now turn our attention to online algorithms designed specifically for grid scheduling. In an environment where neither the resources availability nor the number of jobs to be scheduled are known beforehand, exploiting inter-task dependencies is the only way to efficiently allocate resources and to reduce the schedule length of the job.

Grid computing is relatively new area and algorithms and techniques for utilizing its full potential are continually under development. Specialized resource management architectures (RMA) exist in grid domain to schedule tasks and manage the allocation of heterogeneous resources. Three main resource management architectures that address the needs of grid environment are Globus [19], Condor[20] and Legion[21]. The objective of scheduling algorithms employed in these RMAs is limited to matching only. Complex objective functions such as makespan of the job, or resource utilization are not considered in scheduling algorithms implemented in these architectures. Meta-Scheduler in Globus, for example, resort to reserving all the resources required by the job in advance without considering any interdependence among the tasks. Condor on the other hand handles a single task at a time. In Condor, a task only becomes available when all its predecessors have been processed. Both approaches have their positive as well as negative aspect. In first approach, the algorithm provides the guarantee that all the required resources will be available when needed but may end-up reserving more resources than actually needed. On the other hand Condor's approach is resource efficient, but its lack of reservation mechanism can hinder job completion on time.

In addition to default RMA scheduling algorithms, researchers have proposed pluggable scheduling modules to extend the scheduling capabilities of the resource manager. These scheduling algorithms address specific areas in grid domain such as scheduling of coarse grained tasks [22], master-worker paradigm [23] and parameter-sweep applications [24]. Among them scheduling of coarse-grained precedence constrained tasks is most relevant to our work. Multi-processing environment, such as presented in the aforementioned work, does not provide any guarantee of resource availability, as some remote job may not get any processor time under heavy local load. Our proposed model assumes a different approach in which remote jobs are allowed to run only when resources are idle. This assumption is in line with popular commodity-grid practices employed by various resource intensive computation projects such as Seti@home [25]. Assuming static set of resources is another major drawback of this approach, a case which is not true for grid in general and public-resource computing in particular.

4. Management Architecture for Resource Services (MARS)

Before describing the scheduling approach it would be beneficial to outline the overall working of MARS and its interaction with the outside world. A job is described in the form of a Task Interaction Graph (TIG), in which each vertex represents a task. Every task has associated resource requirements. We term such a graph as resource requirements graph (RRG). MARS would accept jobs that need to be scheduled from a client and "bid" for it based upon the earliest it can complete the job while meeting the resource requirements. The client would relay the job to the site which has the earliest finishing time. When making allocation decisions, MARS has two objectives insight. First, to minimize the completion time and second, to allocate minimum number of resources. These two parameters guarantee efficiency and ability to serve more jobs.

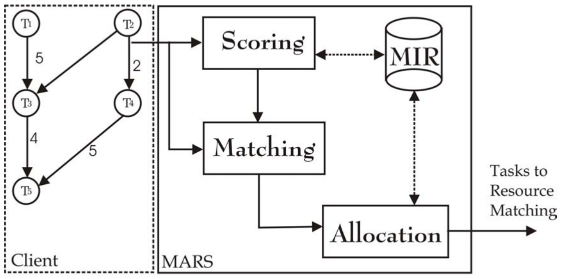


Figure 1: Resource Negotiation Process in MARS.

4.1. Resource Allocation and Task Scheduling

Resource negotiation algorithm in MARS consists of three phases namely Scoring, Matching and Allocation. Scoring algorithm takes resource requirements and generate a list of available resources that match these requirements. Furthermore these matched resources are assigned a score based upon their closeness to required value. Given TIG and list of available "ranked" resources, matching phase identify potential candidates for task allocation. Finally, allocation decision is made by evaluating proximity of these potential matches, in order to ensure minimum network latency between the interacting tasks. Figure 1 is a high level diagram depicting interaction among the three components.

4.2. Match Scoring Scheme

Purpose of scoring (or ranking) process is to identify a pool of potential hosts that match task requirements with the help of MARS Information Repository (MIR). Host suitability for running a task is determined by the score host carries. Degree of weakness, or strongness, of a match is calculated as follows. First of all resources are evaluated for the hard requirements such as operating system and instruction set. The resulting hosts are ranked using the following approach. Assume that a required resource N for a task is stated as $N, \leq N_{pref}, \geq N_{low}$; Where N_{pref} is the preferred valued of resource and N_{low} represents the lowest acceptable value for that resource. Say N is matched with value N_{match} which lies between N_{pref}, and N_{low}. The variation from the preferred value and the maximum variation can be calculated as:

$$N_{var} = N_{pref} - N_{match}$$
$$N_{maxvar} = N_{pref} - N_{low}$$

The score of the match will be:

$$N_{score} = (N_{var} / N_{maxvar}) * 100/10$$

This normalization will yield a score ranging from 0 to 10 for an individual resource match, where 0 indicating a perfect match. A node's score can also be a weighted average of the required resources. This need can arise in cases when some resources are more important than others. Resource matching algorithm will return a set of resources for each task, along with the score.

4.3. Resource Matching

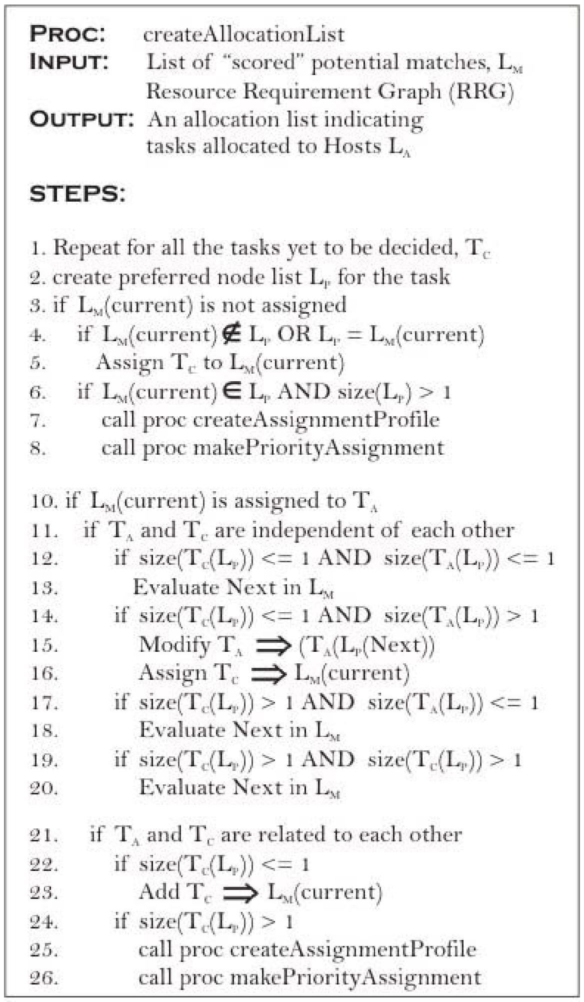
Once potential candidates for running the tasks are identified, next step is to match tasks with hosts. Purpose of resource matching phase is to schedule tasks while satisfying task's resource requirements and keeping number of hosts minimum. Before explaining the algorithm itself, we like to introduce relationship among tasks which will help understanding allocation decisions. Assuming X and Y are two arbitrary tasks in the task graph, X and Y can have either of the following relation with each other.

- **Direct Parent:** Node X is direct parent of Y if there is a directed edge from X to Y. In this case task Y can start directly after task X finishes, given that only parent of Y is X.

- **Indirect Parent:** Node X is indirect parent to node Y if there is no direct edge between them, but they connect with each other through intermediate nodes. Task Y can not start until both its direct and indirect parents finish execution.
- **Independent:** Node X and Y does not have any direct on indirect edge between them and thus their completion is independent of each other.

Each of the above relationship determines allocation of a host to a task. Generally speaking independent tasks can run in parallel and thus should be assigned to different hosts, whereas in child-parent relationship children can be assigned to same host parent is assigned to, if the same host meets requirements of both parent and child tasks. Matching algorithm itself is presented in Listing 1 and is described below.

The main allocation algorithm parses through a list of potential hosts and chooses the best possible host for running the task while keeping the number of host utilized minimum. Altogether task allocation algorithm deals with two main cases.



```
PROC:    createAllocationList
INPUT:   List of "scored" potential matches, $L_M$
         Resource Requirement Graph (RRG)
OUTPUT:  An allocation list indicating
         tasks allocated to Hosts $L_A$

STEPS:

1. Repeat for all the tasks yet to be decided, $T_C$
2. create preferred node list $L_P$ for the task
3. if $L_M$(current) is not assigned
4.    if $L_M$(current) $\notin L_P$ OR $L_P = L_M$(current)
5.       Assign $T_C$ to $L_M$(current)
6.    if $L_M$(current) $\in L_P$ AND size($L_P$) > 1
7.        call proc createAssignmentProfile
8.        call proc makePriorityAssignment

10. if $L_M$(current) is assigned to $T_A$
11.    if $T_A$ and $T_C$ are independent of each other
12.       if size($T_C(L_P)$) <= 1 AND size($T_A(L_P)$) <= 1
13.          Evaluate Next in $L_M$
14.       if size($T_C(L_P)$) <= 1 AND size($T_A(L_P)$) > 1
15.          Modify $T_A \Rightarrow (T_A(L_P(Next))$
16.          Assign $T_C \Rightarrow L_M$(current)
17.       if size($T_C(L_P)$) > 1 AND size($T_A(L_P)$) <= 1
18.          Evaluate Next in $L_M$
19.       if size($T_C(L_P)$) > 1 AND size($T_C(L_P)$) > 1
20.          Evaluate Next in $L_M$

21.    if $T_A$ and $T_C$ are related to each other
22.       if size($T_C(L_P)$) <= 1
23.          Add $T_C \Rightarrow L_M$(current)
24.       if size($T_C(L_P)$) > 1
25.          call proc createAssignmentProfile
26.          call proc makePriorityAssignment
```

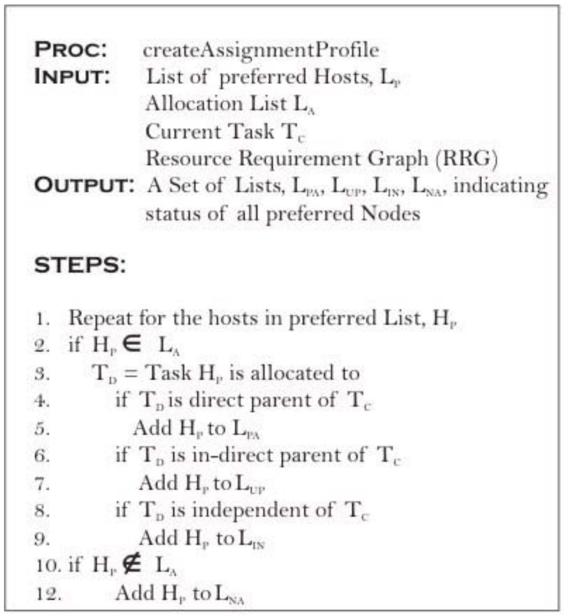
Listing 1 : Main Allocation Algorithm

Case 1 - *Host being evaluated for the current task is not assigned to any task*: The host can be among the best possible ($\in$ Preferred List, L_P) or can be just a potential match ($\notin$ Preferred List, L_P but $\in$ Potential Match List L_M; such that $L_P \subseteq L_M$). A preferred node meets certain threshold score requirement for any task. In cases when host does not belong to L_P or is the only preferred host in the list (Host $\subset L_P$), current task gets the host. If the preferred list has more than one hosts for the current task (Host $\subseteq L_P$) then all the hosts are profiled according to their relationship with other tasks. This step is achieved by procedure *createAssignmentProfile*, by sorting all the preferred nodes into four categories; assigned to parent task, assigned to indirect parent task, assigned to independent task, and not-assigned. Profiling of nodes will help us minimize the number of allocated hosts, e.g. if we find a host in preferred list that is allocated to parent task, then it is best to assign child to the same host, as this will minimize not only the host numbers but also saves us the communication overhead. Following is the decision rationale for the priority assignment of tasks.

- If possible task should be assigned to host parent task is assigned to. This will reduce number of hosts utilized and eliminate inter-task communication overhead between parent and child tasks. (As communication cost of two tasks running at the same host is assumed to be zero)
- Second priority goes to host which is running a task which is indirect parent of the current host. This will only reduce number of hosts utilized.
- Hosts that are not yet assigned come third. Making such assignment will add-up to total number of hosts used, and will not reduce any communication cost.
- Fourth priority goes to host that is assigned to tasks that are independent of the current task. This will add to completion time of the job as a whole and thus has the least priority.

These priorities are handled in part by procedures, *createAssignmentProfile (listing 2)* and *makePriorityAssignment*. Case 1 is handled by steps 3 – 8 of the main algorithm.

Case 2 - *Host being evaluated for the current task(T_C) is assigned to a task(T_A)*: If the required host is allocated to another task, then the decision relies on relationship between these two tasks. If two tasks are related (direct or indirect, parent-child relationship) then we again resort to



```
PROC:   createAssignmentProfile
INPUT:  List of preferred Hosts, $L_P$
        Allocation List $L_A$
        Current Task $T_C$
        Resource Requirement Graph (RRG)
OUTPUT: A Set of Lists, $L_{PA}$, $L_{UP}$, $L_{IN}$, $L_{NA}$, indicating
        status of all preferred Nodes

STEPS:

1.  Repeat for the hosts in preferred List, $H_P$
2.  if $H_P \in L_A$
3.      $T_D$ = Task $H_P$ is allocated to
4.          if $T_D$ is direct parent of $T_C$
5.              Add $H_P$ to $L_{PA}$
6.          if $T_D$ is in-direct parent of $T_C$
7.              Add $H_P$ to $L_{UP}$
8.          if $T_D$ is independent of $T_C$
9.              Add $H_P$ to $L_{IN}$
10. if $H_P \notin L_A$
12.         Add $H_P$ to $L_{NA}$
```

Listing 2 : Procedure to determine relationship among candidate hosts

calling the sub-procedures to evaluate the best possible allocation from preferred nodes. If two tasks are independent of each other, then only way we can assign current task the host being evaluated H_E is if we can make T_A leave H_E. T_A can only leave H_E if it still have more hosts in its preferred lists L_P. In such case, T_A will take-up the next host in preferred list, and H_E will be assigned to T_C. If this is not possible, then T_C will proceed to evaluate next host in its match list. Case 2 is handled by steps 10 to 26 of the main algorithm.

Resource Matching – Example Run

An example run of a job consisting of five tasks is shown in figure 2. We use this example to explain the working of the algorithm and the primary measuring criteria for the results.

Potential matches are generated from 10 hosts (203.135.3.0 – 203.135.3.10) and maximum ranked hosts for any tasks for this particular example were five. For a simple assignment algorithm the allocation was easy, because resource broker was able to match every task with a distinct host that met the resource requirements. Following simple allocation, task 0 was matched with host "203.135.3.0," task 1 to host "203.135.3.4," task 2 to host "203.135.3.7," task 3 to host "203.135.3.3" and task 4 to host "203.135.3.5". Interesting thing to note here is that first host task 4 was matched with is "203.135.3.4" but as this host is already assigned to task 1, the algorithm has to look for more options. Resource matching algorithm in MARS however was able to deduct following two important observations from the task graph.

- Task 1, 2 and 4 are dependent on each other, i.e. Task 2 can not start until Task 1 finishes, and Task 4 can not start until Task 2 finishes
- Task 1, 2 and 4 have a host in common, which perfectly matches all requirements put forward by these tasks, i.e. 203.135.3.4.

Based on these observations MARS allocation algorithm created the assignment shown in Figure 2(c).

Task 0 and Task 3 were independent and/or they had different resource requirements and thus have their individual host mapping. There are couple of things to note in this assignment. First we have minimized number of hosts used, achieving efficiency of 60%; secondly we have also reduced the critical path of the job. A critical path is defined as the path which dominates overall execution of the workload. In the original task graph, the critical path involved task 0, 1, 2, and 3. Giving the communication cost of 4 (0 to 1), 8(1 to 2) and 8(2 to

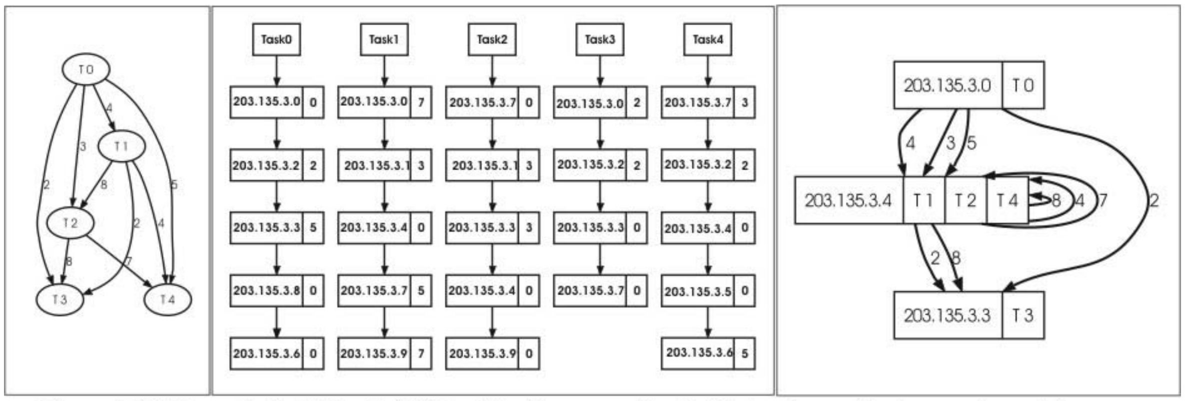


Figure 2: (a) Example Task Graph (b) Matching Resources for the Tasks along with degree of matching (c)Assignment of Tasks to Hosts made by MARS Hosts Allocation Algorithm

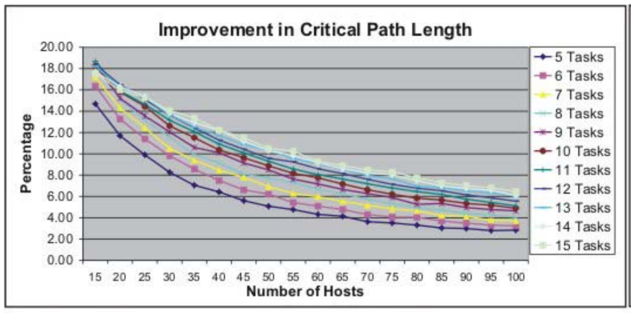


Figure 3: **MARS** - Improvement in Critical Path Length

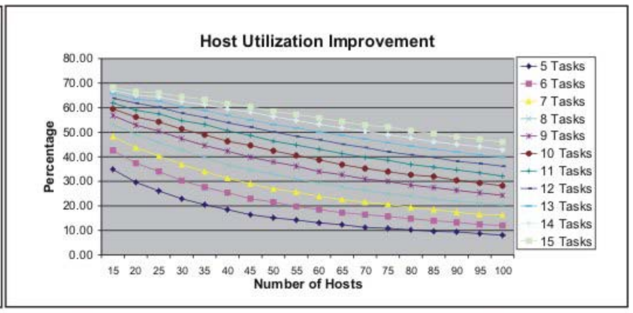


Figure 4: **MARS** - Improvement in Host Utilization

3) plus the cost of executing the tasks themselves which are assumed to be taking single unit of time at the moment. This adds up to 24. In MARS assignment, though critical path remained the same, the communication cost of task 1 to 2 is no longer there as both these tasks are assigned to the same host. This reduces the critical path to 16. These two attributes namely minimized number of hosts used and critical path reduction are our main contribution to the resource allocation paradigm is concerned.

Results and Discussion

There are three major criteria for evaluating performance of MARS resource allocation and task scheduling component. Namely, Makespan of the job, number of allocated resources, and time taken by the scheduling and allocation algorithm. There is a number of different parameters that are considered while designing the experiments. Number of tasks in the task graph, number of hosts present in the grid domain, and the connectivity of the task graph are among the major ones. The first two parameters represent the state of the grid at the arrival of a job and client requests. Task graph connectivity serves as a normalization parameter, so that results are not biased towards certain type of task graphs. Considering all the parameters, result for a single TIG represents 62475 runs. The resulting values are compared against a resource management scheme with advance reservation mechanism such as Globus. In the context of TIG, critical path is the most time consuming (longest) path in the graph. This includes the execution time of tasks and time involved in exchanging messages between the tasks. Time of the critical path establishes the lower bound for job completion, i.e. a job can not complete earlier than the finish time of critical path. Thus reduction in critical path effectively translates into overall shorter makespan of the job itself. Figure 4 and 5 shows relative percentage improvement. The improvement in critical path depends directly on how many tasks are clustered together. Therefore there is a sharp decline in critical path reduction when hosts utilization decreases. Host utilization is dependent upon what hosts parent task picks. Therefore when number of hosts increases, and so does the probability to find perfect matching hosts, host utilization decreases. This trend emphasize the need for forward looking mechanism which also takes into account hosts needed by the child tasks when making host allocation decision for a task.

Algorithm's running time show a very promising picture, albeit for a smaller set of tasks. For 15 tasks algorithm was able to make allocation decision in 180 milliseconds.

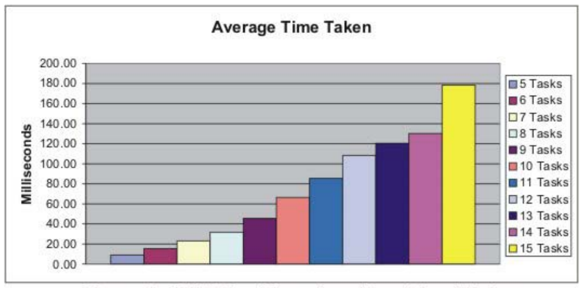


Figure 5: **MARS** - Allocation Algorithm Timings

5. Future Directions

Preliminary results presented in this paper validate our hypothesis that considering task interdependence in grid environment can produce considerable improvement in job's makespan and resource utilization. The scheduling algorithm we have presented in this paper assumes tasks of unit time length. In future we plan to relax this restriction by taking into account variable length tasks.

Besides task matching, scheduling in grid poses additional challenges [26]. Fault tolerance is crucial for systems that are composed of resources prone to failures. Impact of the resource failure varies depending upon the current state of the resource. If the resource is currently not running any task then marking the resource as unavailable would be sufficient to prevent its assignment to a task. In case resource was executing a task, than failure recovery procedure should recover completed portion of the task and find an alternate resource to complete the task. QoS is another important issue when designing a grid resource management system. In some cases a task may be under-matched because of absence of the ideal set of resources needed to execute the tasks. Under such circumstances a resource allocation re-evaluation mechanism can help locating the ideal match that becomes available at later stages of task execution.

References

[1]. S. Hastings, T. Kurc, S. Langella, U. Catalyurek, T. Pan, and J. Saltz. "Image Processing on the Gird: A Toolkit for Building Grid-enabled Image Processing Applications". In 3rd International Symposium on Cluster Computing and the Grid. 2003.

[2]. S. Ludtke, P. Baldwin, and W. Chiu. EMAN: Semiautomated Software for high-resolution single-particle reconstructions. J. Struct. Biology. 128:82-97. 1999.

[3]. G. B. Berriman, et al. Montage: A Grid enabled image mosaic service for the national virtual environments. ADASS XIII ASP Conference Series. 2004.

[4]. D. Fernandez-Baca. Allocating modules to processors in a distributed system. IEEE Transaction on Software Engineering, SE-15(11): 1427 – 1436, Nov. 1989.

[5]. O.H. Ibrarra and C. E. Kim. Heuristic Algorithms for scheduling independent tasks on non-identical processors. Journal of the ACM, 24(2): 280 – 289, Apr. 1977.

[6]. M.R. Gary and D. S. Johnson, Computer and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co, 1979.

[7]. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnoy Kan. Optimization and Approximation in Deterministic sequencing and Scheduling: A Survey. Annals of Discrete Mathematics. 5:287 – 326, 1979.

[8]. T. Cassavant and J. A. Juhl, "Taxonomy of Scheduling in General Purpose Distributed Memory Systems," Transaction on Software Engineering, SE-14(2). 141- 154. 1988.

[9]. V. J. Rayward-Smith. UET scheduling with inter-processor communication delays. Technical Report SYS-C86-06. School of Information Systems. University of East Anglia. Norwich. 1986.

[10]. M. R. Garey and D. S. Johnson. Strong NP-Completeness results: motivation, examples, and implications. Journal of the ACM. 25(3): 499 – 508. 1978

[11]. J. D. Ullman. NP Complete Scheduling Problems. Journal of Computer and System Sciences. 10: 384 – 393. 1975

[12]. W. Kubiak. Exact and Approximate Algorithms for scheduling unit time tasks with tree-like precedence constraints. EURO IX – TIMS XXVIII Paris, page 195. 1988

[13]. W. Kubiak, B. Penzabd D. Trystram. Scheduling Chains on Uniform processors with communication delays. Journal of Scheduling. 5(6): 459 – 476, 2002.

[14]. T. Braun, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A. I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hengsen, and R.F. Freund. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. Proceedings of the International Heterogenous Computing Workshop (HCW, 99). Pp. 15-29. April 1999.

[15]. Y. Kwok and I. Ahmad, "Benchmarking and Comparison of the Task Graph Scheduling Algorithms," Journal of Parallel and Distributed Processing. March 1999.

[16]. H. El-Rewini, H. Ali and T. Lewis, "Task Scheduling in Multiprocessing Systems," IEEE Computer. Dec. 1995.

[17]. Annie S. Wu, Han Yu, Shiyuan Jin, Kuo-Chi Lin, and Guy Schiavone. "An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling". IEEE Transactions on Parallel and Distributed Systems, Vol. 15, No. 9. September 2004.

[18]. Haluk Topcuoglu, Salim Hariri, Min-You Wu, "Performance Effective and Loq-Complexity Task Scheduling for Heterogeneous Computing". IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 3. March 2002.

[19]. http://www.globus.org/alliance/publications/papers.php

[20]. http://www.cs.wisc.edu/condor/publications.html

[21]. http://www.cs.virginia.edu/~legion/papers.html

[22]. Noriyaki Fujimoto and Kenichi Hagihara. Near-Optimal Dynamic Task Scheduling of Precedence Constrained Coarse-Grained Tasks onto a Computational Grid. The 2nd International Symposium on Parallel and Distributed Computing (ISPDC 2003), pp.80-87, IEEE Press, Ljubljana, Slovenia, October 16-18, 2003

[23]. Cyril Banino, Olivier Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, Yves Robert. Scheduling Strategies for Master-Slave Tasking on Heterogenous Processor Platforms. IEEE Transactions on Parallel and Distributed Systems, Vol. 15, No. 4. April 2004.

[24]. Sivakumar Viswanathan, Bharadwaj Veervalli, Dantong Yu, and Thomas G. Robertazzi. Design and Analysis of a Dynamic Scheduling Strategy with Resource Estimation for Large-Scale Grid Systems. Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04).

[25]. http://setiathome.ssl.berkeley.edu/

[26]. Shoukat Ali, Tracy D. Braun, Howard J. Siegel, Anthony A. Maciejewski, Noah Beck, Ladislau Boloni, Muthucumaru Mahehwaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys and Bin Yao, "Characterizing Resource Allocation Heuristics for Heterogeneous Computing Systems". Computer Architecture, Elsevier, 2004