A Unified Resource Scheduling Framework for Heterogeneous Computing Environments

Ammar H. Alhusaini * and Viktor K. Prasanna*
Department of EE-Systems, EEB 200C
University of Southern California
Los Angeles, CA 90089-2562
Ph: (213) 740-4483
{ammar + prasanna}@usc.edu

C.S. Raghavendra
The Aerospace Corporation
P. O. Box 29257
Los Angeles, CA 90009
Ph: (310) 336-1686
raghu@aero.org

Abstract

A major challenge in Metacomputing Systems (Computational Grids) is to effectively use their shared resources, such as compute cycles, memory, communication network, and data repositories, to optimize desired global objectives. In this paper we develop a unified framework for resource scheduling in metacomputing systems where tasks with various requirements are submitted from participant sites. Our goal is to minimize the overall execution time of a collection of application tasks. In our model, each application task is represented by a Directed Acyclic Graph (DAG). A task consists of several subtasks and the resource requirements are specified at subtask level. Our framework is general and it accommodates emerging notions of Quality of Service (QoS) and advance resource reservations. In this paper, we present several scheduling algorithms which consider compute resources and data repositories that have advance reservations. As shown by our simulation results, it is advantageous to schedule the system resources in a unified manner rather than scheduling each type of resource separately. Our algorithms have at least 30% improvement over the separated approach with respect to completion time.

1. Introduction

With the improvements in communication capability among geographically distributed systems, it is attractive to use diverse set of resources to solve challenging applications. Such Heterogeneous Computing (HC) systems [12, 17] are called *metacomputing* systems [26] or *computational grids* [8]. Several research projects are underway, including for example, MSHN [22], Globus [13], and Legion [19], in which the users can select and employ resources at different domains in a seamless manner to execute their applications. In general, such metacomputing systems will have compute resources with different capabilities, display devices, and data repositories all interconnected by heterogeneous local and wide area networks. A variety of tools and services are being developed for users to submit and execute their applications on a metacomputing system.

A major challenge in using metacomputing systems is to effectively use the available resources. In a metacomputing environment, applications are submitted from various user sites and share system resources. These resources include compute resources, communication resources (network bandwidth), and data repositories (file servers). Programs executing in such an environment typically consist of one or more subtasks that communicate and cooperate to form a single application. Users submit jobs from their sites to a metacomputing system by sending their tasks along with Quality of Service (QoS) requirements.

Task scheduling in a distributed system is a classic problem (for a detailed classification see [5, 6]). Recently, there have been several works on scheduling tasks in metacomputing systems. Scheduling independent jobs (meta-tasks) has been considered in [2, 11, 14]. For application tasks represented by Directed Acyclic Graphs (DAGs), many dynamic scheduling algorithms have been devised. These include the Hybrid Remapper [20], the Generational algorithm [9], as well as others [15, 18]. Several static algorithms for scheduling DAGs in metacomputing systems are described in [16, 23, 24, 25, 27]. Most of the previous algorithms focus on compute cycles as the main resource. Also, previous DAGs scheduling algorithms assume that a subtask receives all its input data from its predecessor subtasks. Therefore, their scheduling decisions are based on machine performance for the subtasks and the cost of receiving input

*Supported by the DARPA/ITO Quorum Program through the Naval Postgraduate School under subcontract number N62271-97-M-0931.

data from predecessor subtasks only.

Many metacomputing applications need other resources, such as data repositories, in addition to compute resources. For example, in data-intensive computing [21] applications access high-volume data from distributed data repositories such as databases and archival storage systems. Most of the execution time of these applications is in data movement. These applications can be computationally demanding and communication intensive as well [21]. To achieve high performance for such applications, the scheduling decisions must be based on all the required resources. Assigning a task to the machine that gives its best execution time may result in poor performance due to the cost of retrieving the required input data from data repositories. In [4], the impact of accessing data servers on scheduling decisions has been considered in the context of developing an AppLes agent for the Digital Sky Survey Analysis (DSSA) application. The DSSA AppLes selects where to run a statistical analysis according to the amount of required data from data servers. However, the primary motivation was to optimize the performance of a particular application.

In this paper we develop a unified framework for resource scheduling in metacomputing systems. Our framework considers compute resources as well as other resources such as the communication network and data repositories. Also, it incorporates the emerging concept of *advance reservations* where system resources can be reserved in advance for specific time intervals. In our framework, application tasks with various requirements are submitted from participant sites. An application task consists of subtasks and is represented by a DAG. The resource requirements are specified at the subtask level. A subtask's input data can be data items from its predecessors and/or data sets from data repositories. A subtask is ready for execution if all its predecessors have completed, and it has received all the input data needed for its execution. In our framework, we allow for input data sets to be replicated, i.e., the data set can be accessed from one or more data repositories. Additionally, a task can be submitted with QoS requirements, such as needed compute cycles, memory, communication bandwidth, maximum completion time, priority, etc. In our framework, sources of input data and the execution times of the subtasks on various machines along with their availability are considered simultaneously to minimize the overall completion time.

Although our unified framework allows many factors to be taken into account in resource scheduling, in this paper, to illustrate our ideas, we present several heuristic algorithms for a resource scheduling problem where the compute resources and the data repositories have advance reservations. These resources are available to schedule subtasks only during certain time intervals as they are reserved (by other users) at other times. QoS requirements such as deadlines and priorities will be included in future algorithms. The objective of our resource scheduling algorithms is to minimize the overall completion time of all the submitted tasks.

Our research is a part of the MSHN project [22], which is a collaborative effort between DoD (Naval Postgraduate School), academia (NPS, USC, Purdue University), and industry (NOEMIX). MSHN (Management System for Heterogeneous Networks) is designing and implementing a Resource Management System (RMS) for distributed heterogeneous and shared environments. MSHN assumes heterogeneity in resources, processes, and QoS requirements. Processes may have different priorities, deadlines, and compute characteristics. The goal is to schedule shared compute and network resources among individual applications so that their QoS requirements are satisfied. Our scheduling algorithms, or their derivatives, may be included in the Scheduling Advisor component of MSHN.

This paper is organized as follows. In the next section we introduce our unified resource scheduling framework. In Section 3, we present several heuristic algorithms for solving a general resource scheduling problem which considers input requirements from data repositories and advance reservations for system resources. Simulation results are presented in Section 4 to demonstrate the performance of our algorithms. Finally, Section 5 gives some future research directions.

2. The Scheduling Framework

2.1. Application Model

In the metacomputing system we are considering, n application tasks, $\{T_1, \ldots, T_n\}$, compete for computational as well as other resources (such as communication network and data repositories). Each application task consists of a set of communicating subtasks. The data dependencies among the subtasks are assumed to be known and are represented by a Directed Acyclic Graph (DAG), $G = (V, E)$. The set of subtasks of the application to be executed is represented by V=$\{v_1, v_2, \ldots, v_k\}$ where $v_k \geq 1$, and E represents the data dependencies and communication between subtasks. e_{ij} indicates communication from subtask v_i to v_j, and $|e_{ij}|$ represents the amount of data to be sent from v_i to v_j. Figure 1 shows an example with two application tasks. In this example, task 1 consists of three subtasks, and task 2 consists of nine subtasks.

In our framework, QoS requirements are specified for each task. These requirements include needed compute cycles, memory, communication bandwidth, maximum completion time, etc. In our model, a subtask's input data can be data items from its predecessors and/or data sets from data repositories. All of a subtask's input data (the data items and the data sets) must be retrieved before its execution. After

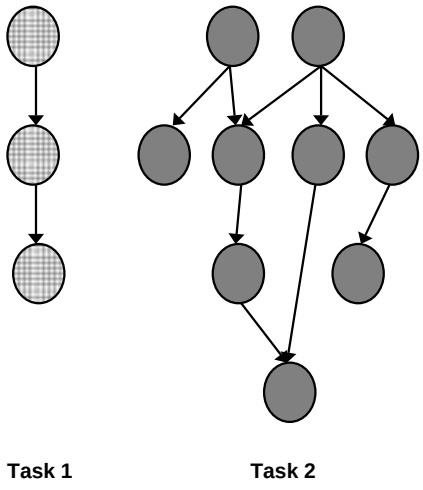

Figure 1. Example of application tasks

a subtask's completion, the generated output data may be forwarded to successor subtasks and/or written back to data repositories.

In some applications, a subtask may contain sub-subtasks. For example, Adaptive Signal Processing (ASP) applications are typically composed of a sequence of computation stages (subtasks). Each stage consists of a number of identical sub-subtasks (i.e., FFT's, QR decompositions, etc.). Each stage repeatedly receives its input from the previous stage, performs computations, and sends its output to the next stage.

2.2. System Model

The metacomputing system consists of m heterogeneous machines, $M = \{m_1, m_2, \ldots, m_m\}$, and f file servers or data repositories, $S = \{s_1, s_2, \ldots, s_f\}$. We assume that an estimate of the execution time of subtask v_i on machine m_j is available at compile-time. These estimated execution times are given in matrix ECT. Thus, $ECT(i, j)$ gives the estimated computation time for subtask i on machine j. If subtask v_i cannot be executed on machine m_j, then $ECT(i, j)$ is set to infinity.

System resources may not be available over some time intervals due to advance reservations. Available time intervals for machine m_j are given by $MA[j]$. Available time intervals for data repository s_j are given by $SA[j]$. Matrices TR and L give the message transfer time per byte and the communication latency between machines respectively. Matrices $Data\_TR$ and $Data\_L$ specify the message transfer time per byte and the communication latency between the data repositories and the machines, respectively. $DataSet[i]$ gives the amount of input data sets needed from data repositories for subtask v_i. In systems with multiple copies of data sets, one or more data repository can provide the required data sets for that subtask.

2.3. Problem Statement

Our goal is to minimize the overall execution time for a collection of applications that compete for system resources. This strategy (i.e., optimizing the performance of a collection of tasks as opposed to that of a single application) has been taken by SmartNet [11] and MSHN [22]. On the other hand, the emphasis in other projects such as AppLes [3] is to optimize the performance of an individual application rather than to cooperate with other applications sharing the resources. Since multiple users share the resources, optimizing the performance of an individual application may dramatically affect the completion time of other applications.

We now formally state our resource scheduling problem.

Given:

- A Metacomputing system with m machines and f data repositories,
- Advance reserved times for system resources as given by MA and SA,
- n application tasks, $\{T_1, \ldots, T_n\}$,where each application is represented by a DAG,
- Communication latencies and transfer rates among the various resources in matrices TR, $L, Data\_TR$, and $Data\_L$,
- Subtasks execution times on various machines in matrix ETC, and
- Amount of input data sets needed from data repositories for each subtask v_i as given by $DataSet[i]$.

Find a schedule to

$$Minimize\ \{\max_{j=1}^{n}\ [Finish\ Time(T_j)]\ \},$$

where the schedule determines, for each subtask, the start time and the duration of all the resources needed to execute that subtask.

Subject to the following constraints:

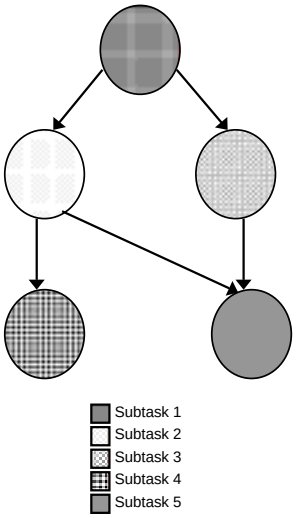

Figure 2. Application DAG for the example in Sec. 2.4

	m_1	m_2	m_3
V_1	5	4	8
V_2	20	5	3
V_3	6	10	4
V_4	10	4	2
V_5	∞	6	5

Table 1. Subtask execution times

- A subtask can execute only after all its predecessors have completed, all input data items have been received from its predecessors, and the input data sets have been retrieved from one of the data repositories,
- Preserve all advance resource reservations,
- Only one subtask can execute on any machine at any given time, and
- At most one subtask can access any data repository at any given time.

2.4. Separated Scheduling Vs. Unified Scheduling

Many scheduling methods exist in the literature for scheduling application DAGs on compute and network resources. They do not consider data repositories. With the inclusion of data repositories, one can obtain schedules for compute resources and data repositories independently and

	m_1	m_2	m_3
S_1	5	6	6
S_2	1	4	3
S_3	4	1.5	5

Table 2. Transfer costs (time units/data unit)

Subtask	Amount of the Input Data Set	Data Repository Choices
V_1	3 units	S_1 or S_2
V_2	10 units	S_2 or S_3
V_3	2 units	S_1 or S_3
V_4	1 unit	S_1 or S_2
V_5	5 units	S_3

Table 3. Input requirements for the subtasks

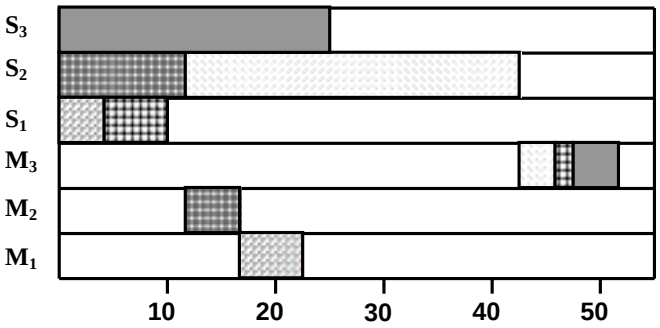

Figure 3. Separated scheduling (machines first)

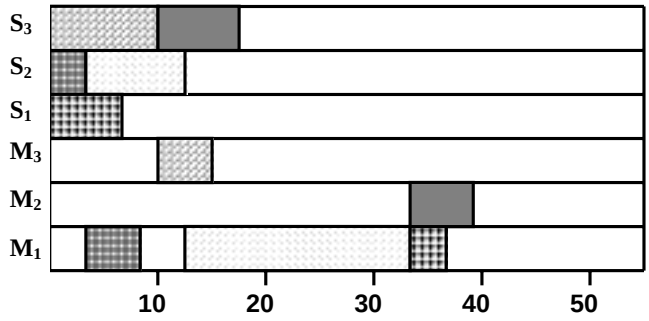

Figure 4. Separated scheduling (data repositories first)

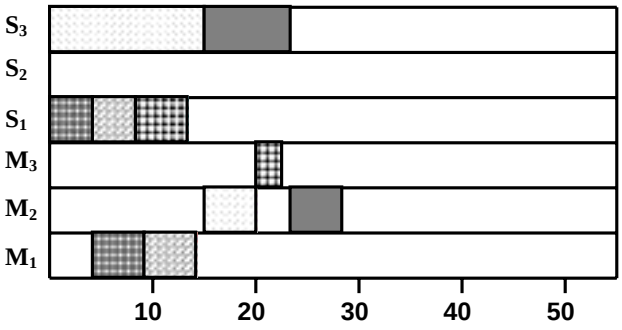

Figure 5. Unified scheduling

combine the schedules. In this section we show with a simple example, that this separated approach is not efficient with respect to completion time.

Figure 2 shows the DAG representation for an application task with 5 subtasks. In this example, we assume a fully connected system with 3 machines and 3 data repositories (file servers). The subtask execution times (in time units) are given in Table 1. Table 2 gives the the cost (in time units) for transferring one data unit from any data repository to any machine. We assume that each subtask needs an input data set, which can be retrieved from one or more data repositories as given in Table 3.

In this example, we are using a simple list scheduling algorithm called the Baseline Algorithm. This algorithm has been described in [20, 27]. The baseline algorithm is a fast static algorithm for mapping DAGs in HC environments. It partitions the subtasks in the DAG into blocks (levels) using an algorithm similar to the level partitioning algorithm which will be described in Section 3.1. Then all the subtasks are ordered such that the subtasks in block k come before the subtasks in block b, where $k < b$. The subtasks in the same block are sorted in descending order based on the number of descendents of each subtask (ties are broken arbitrarily). The subtasks are considered for mapping in this order. A subtask is mapped to the machine that gives the minimum completion time for that particular subtask. Since the original algorithm does not account for the data repositories, we implemented a modified version of the algorithm. In the modified version, the algorithm chooses a data repository that gives the best retrieving time of the input data set.

The schedule based on the separated approach, when scheduling the machines first, is shown in Figure 3. The completion time of this schedule is 52 time units. For this case, we map the application subtasks to the machines as they are the only resources in the system. Then for each subtask we choose the data repository that gives the best retrieving (delivery) time of the input data set to the previously mapped machine for this subtask in order to minimize its completion time. The completion time of the schedule based on the separated approach, when scheduling the data repositories first, is 39 time units as shown in Figure 4. For this case, we map the application subtasks to the data repositories as they are the only system resources. Then for each subtask we choose the machine that gives the best completion time for that subtask when using the previously mapped data repository to get the required data set for this subtask. Figure 5 shows the schedule based on the unified approach. The completion time of the unified scheduling is 28.5 time units. In the unified approach, we map each subtask to a machine and data repository at the same time in order to minimize its completion time.

The previous example shows clearly that the scheduling based on the separated approach is not efficient with respect to completion time. Further, with advance reservations, separated scheduling can lead to poor utilization of resources when one type of resource is not available while others are available.

3. Resource Scheduling Algorithms

In this section, we develop static (compile-time) heuristic algorithms for scheduling tasks in a metacomputing system where the compute resources and the data repositories have *advance reservations*. These resources are available to schedule subtasks only during certain time intervals as they are reserved (by other users) at other times. Although our framework incorporates the notion of QoS, the algorithms we present in this paper do not consider QoS. We are currently working on extending our scheduling algorithms to

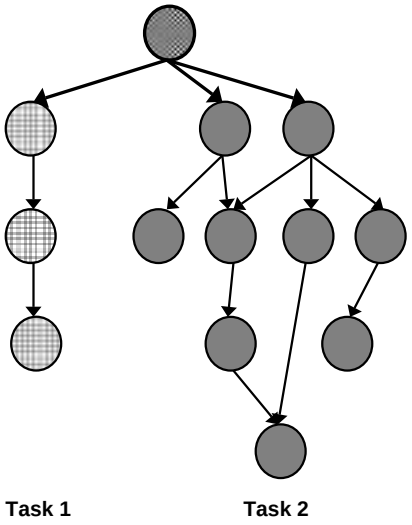

Figure 6. Combined DAG for the tasks in Fig. 1

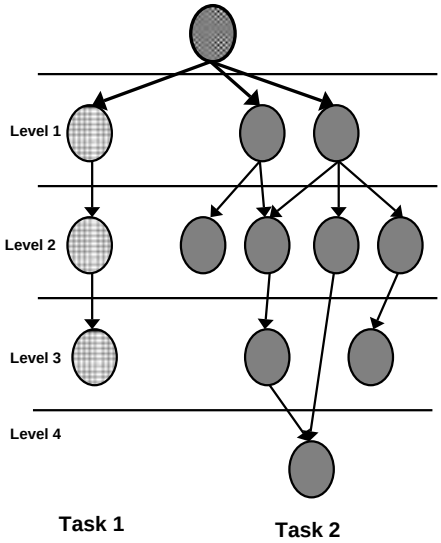

Figure 7. Level partitioning for the combined DAG in Fig. 6

consider QoS requirements such as deadlines, priorities, and security.

As in state-of-the-art systems, we assume a central scheduler with a given set of static application tasks to schedule. With static applications, the complete set of task to be scheduled is known a priori. Tasks from all sites are sent to the central scheduler to determine the schedule for each subtask so that the global objective is achieved. The information about the submitted tasks as well as status of various resources are communicated to the central scheduler. This centralized scheduler will then make appropriate decisions and can achieve better utilization of the resources.

Scheduling in metacomputing systems, even if we schedule based on compute resources only, is known to be NP-complete. One method is based on the well known list scheduling algorithm [1, 16, 23]. In list scheduling, all the subtasks of a DAG are placed in a list according to some priority assigned to each subtask. A subtask cannot be scheduled until all its predecessors have been scheduled. Ready subtasks are considered for scheduling in order of their priorities. In this section, we develop modified versions of list scheduling algorithm for our generalized task scheduling problem with advance resource reservations. Our heuristic algorithms that are based on the list scheduling are of two types – level by level scheduling and greedy approach. In the following, we briefly describe these two types of algorithms.

3.1. Level-By-Level Scheduling

In our framework, application tasks are represented by DAGs where a node is a subtask and the edges from predecessors represent control flow. Each subtask has computation cost, data items to be communicated from predecessor subtasks, and data sets from one or more repositories. A subtask is ready for execution if all its predecessors have completed, and it has received all the input data needed for its execution. To facilitate the discussion of our scheduling algorithms, a hypothetical node is created and linked, with zero communication time links, to the root nodes of all the submitted DAGs to obtain one combined DAG. This dummy node has zero computation time. Figure 6 shows the combined DAG for the two tasks in Figure 1. Now, minimizing the maximum time to complete this combined DAG achieves our global objective.

In level-by-level heuristic, we first partition the combined DAG into l levels of subtasks. Each level contains independent subtasks, i.e., there are no dependencies between the subtasks in the same level. Therefore, all the subtasks in a level can be executed in parallel once they are ready. Level 0 contains the dummy node. Level 1 contains all subtasks that do not have any incident edges originaly, i.e., subtasks without any predecessors in the original DAGs. All subtasks in level l have no successors. For each subtask v_j in level k, all of its predecessors are in levels 0 to $k-1$, and at least one of them in level k-1. Figure 7 shows the levels of the combined DAG in Fig. 6. The combined DAG in this example

```
Level-by-Level Scheduling Algorithm
begin
    Combine all submitted DAGs into one DAG.
    Do level partitioning for the combined DAG.
    For level := 1 to l do
        Set Ready to be the set of all subtasks at this level.
        While Ready is not empty do
            Find FINISH(v_i, m_min, s_min) for all subtasks in Ready, where m_min is
              the machine that gives the minimum completion time for subtask v_i
              if data repository s_min has been used to get the input data set.
            Min-FINISH: Choose the subtask v_k with the minimum completion time.
            Max-FINISH: Choose the subtask v_k with the maximum completion time.
            Schedule subtask v_k to machine m_min and data repository s_min.
            Update MA(m_min) and SA(s_min).
            Remove v_k from Ready.
        end While
    end For
end
```

Figure 8. Pseudo code for the level-by-level scheduling algorithms

has 4 levels.

The scheduler considers subtasks in each level at a time. Among the subtasks in a particular level i, the subtask with the minimum completion time will be scheduled first in the *Min-FINISH* algorithm and the subtask with the maximum completion time is scheduled first in the *Max-FINISH* algorithm. The advance reservations of compute resources and data repositories are handled by choosing the first-fit time interval to optimize the completion time of a subtask.

The idea behind the *Min-FINISH* algorithm, as in algorithm D in [14] and Min-min algorithm in SmartNet [11], is that at each step, we attempt to minimize the finish time of the last subtask in the ready set. On the other hand, the idea in the *Max-FINISH*, as in algorithm E in [14] and Max-min algorithm in SmartNet [11], is to minimize the worst case finishing time for critical subtasks by giving them the opportunity to be mapped to their best resources. The pseudo code for the level-by-level scheduling algorithms is shown in Figure 8.

3.2. Greedy Approach

Since the subtasks in a specific level i of the combined DAG belong to different independent tasks, by scheduling level by level we are creating dependency among various tasks. Further, the completion times of levels of different tasks can vary widely, and the level-by-level scheduling algorithms may not perform well. The idea in the greedy heuristics, *Min-FINISH-ALL* and *Max-FINISH-ALL*, is to consider subtasks in all the levels that are ready to execute in determining their schedule. This will advance execution of different tasks by different amounts and will attempt to achieve the global objective and provide good response times for short tasks at the same time. As before, we consider both the minimum finish time and the maximum finish time of all ready subtasks in determining the order of the subtasks to schedule.

The two greedy algorithms, *Min-FINISH-ALL* and *Max-FINISH-ALL* algorithm, are similar to *Min-FINISH* and *Max-FINISH* respectively. They only differ with respect to the $Ready$ set. In the greedy algorithms, the $Ready$ set may contain subtasks from several levels. Initially, the $Ready$ set contains all subtasks at level 1 from all applications. After mapping a subtask, the algorithms check if any of its successors are ready to be considered for scheduling and add them to $Ready$ set. A subtask cannot be considered for scheduling until all its predecessors have been scheduled.

4. Results and Discussion

For the generalized resource scheduling problem considered above, it is not clear which variation of the list scheduling will perform best. Our intuition is that scheduling subtasks by considering all resource types together will result in bounded suboptimal solutions. In order to evaluate the effectiveness of the scheduling algorithms discussed in Sections 3.1 and 3.2, we have developed a software simulator that calculates the completion time for each of them. The input parameters are given to the simulator as fixed values or as a range of values with a minimum and maximum value.

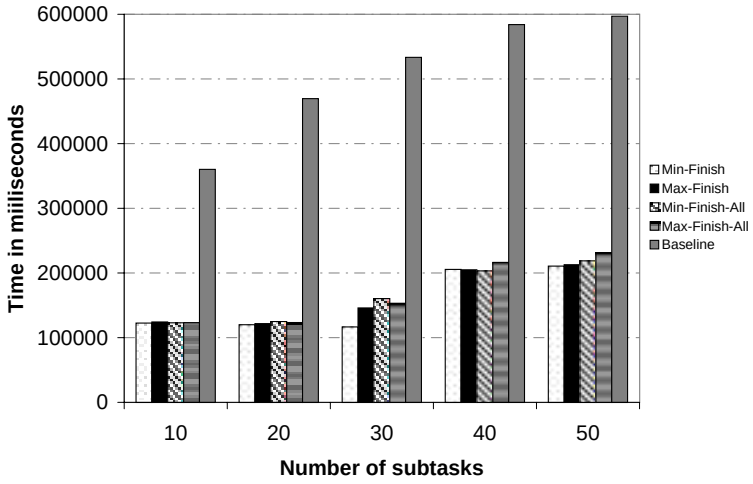

Figure 9. Simulation results for 20 machines and 6 data repositories with varying number of subtasks

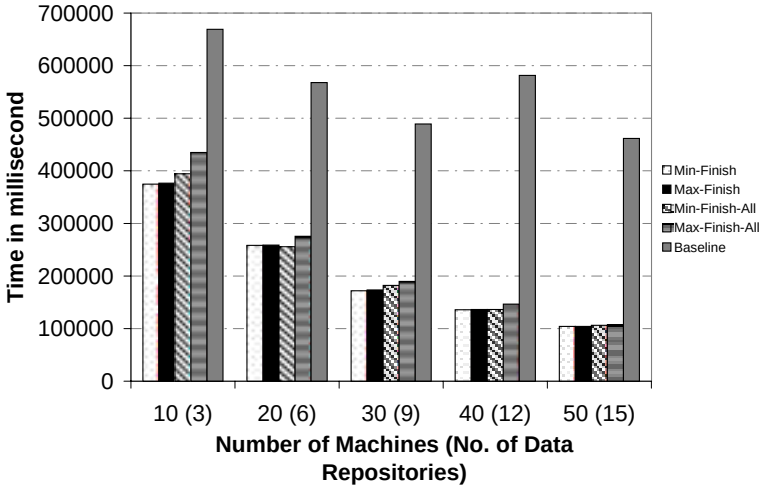

Figure 10. Simulation results for 50 subtasks with varying number of machines and data repositories

Subtask execution times, communication latencies, communication transfer rates, data items amounts, and data sets amounts, are specified to the simulator as range of values. The actual values of these parameters are choosen randomly by the simulator within the specified ranges. The fixed input parameters are the number of machines, the number of data repositories, the number of data items, and the total number of subtasks.

We assume that each task needs an input data set from the data repositories. This data set can be replicated and may be retreived from one or more data repositories. Each compute resource and data repository had several slots blocked at the beginning of the simulation to indicate advance reservations. We compare our scheduling algorithms with separated version of the baseline algorithm discussed in Section 2.4. The simulation results are shown in Figures 9 and 10. In Figure 9, the scheduling algorithms are compared for varying number of subtasks using 20 machines and 6 data repositories. Figure 10 shows a similar comparison for varying number of machines and data repositories with 50 subtasks. Our preliminary results show that all four of our heuristic algorithms seem to have similar performance with relatively uniform task costs. The simulation results clearly show that it is advantageous to schedule the system resources in a unified manner rather than scheduling each type of resource separately. Our scheduling algorithms have at least 30% improvment over the baseline algorithm which use the separated approach.

5. Future Work

This work represents, to the best of our knowledge, the first step towards a unified framework for resource scheduling with emerging constraints that are important in metacomputing. In this paper, we have considered one such requirement of advance reservations for compute resources and data repositories in this paper. We are investigating the question of how advance reservations impact task completion times. That is, in the scheduling, how soon we want to reserve a resource for a subtask to avoid waiting for another resource and/or blocking a different subtask. We are currently working on extending our scheduling algorithms to consider QoS requirements such as deadlines, priorities, and security. We are investigating the mapping of QoS specified at task level to subtasks in our framework.

In our future work we plan to develop scheduling algorithms for dynamic environments with the above mentioned resource requirements. In a dynamic environment, application tasks arrive in a real-time non-deterministic manner. System resources may be removed, or new resources may be added during run-time. Dynamic scheduling algorithms make use of real-time information and require feedback from the system.

References

[1] T. Adam, K. Chandy, and J. Dickson, "A comparison of list schedules for parallel processing systems," *Comm. of the ACM*, 17(12):685-690, Dec. 1974.

[2] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithm is independent of sizable variance in run-time predictions," *7th Heterogeneous Computing Workshop (HCW' 98)*, pp. 79-87, March 1998.

[3] F. Berman and R. Wolski, "Scheduling from the perspective of the application," *5th IEEE International Symposium on High Performance Distributed Computing*, August 1996.

[4] F. Berman, "High-Performance schedulers," in *The Grid: blueprint for new computing infrastructure*, I. Foster and C. Kesselman, ed., Morgan Kaufmann Publishers, San Francisco, CA, 1999, pp. 279-309.

[5] T. Braun et al., "A Taxonomy for describing matching and scheduling heuristics for mixed-machines heterogeneous computing systems," *Workshop on Advances in Parallel and Distributed Systems (APADS)*, West Lafayette, IN, Oct. 1998.

[6] T. Casavant and J. Kuhl, " A Taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Trans. on Software Engineering*, 14(2):141-154, Feb. 1988.

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

[8] I. Foster and C. Kesselman, ed., *The Grid: blueprint for new computing infrastructure*, Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[9] R. Freund, B. Carter, D. Watson, E. Keith, and F. Mirabile, "Generational scheduling for heterogeneous computing systems," *Int'l Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA '96)*, pp. 769-778, Aug. 1996.

[10] R. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multi-user, heterogeneous computing environments with SmarNet," *7th Heterogeneous Computing Workshop (HCW '98)*, pp. 184-199, March 1998.

[11] R. Freund, T. Kidd, D. Hensgen, and L. Moore, "SmartNet: a scheduling framework for heterogeneous computing," *The International Symposium on Parallel Architectures, Algorithms, and Networks*, Beijing, China, June 1996.

[12] R. Freund and H. J. Siegel, "Heterogeneous processing" *IEEE Computer*, 26(6):13-17, June 1993.

[13] Globus Web Page. *http://www.globus.org*.

[14] O. Ibarra and C. Kim, "Heuristic algorithms for scheduling independent tasks on non identical processors." *Journal of The ACM*, 24(2):280-289, April 1977.

[15] M. Iverson and F. Ozguner, "Dynamic, competitive scheduling of multiple DAGs in a distributed heterogeneous environment," *7th Heterogeneous Computing Workshop (HCW' 98)*, pp. 70-78, March 1998.

[16] M. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," *4th Heterogeneous Computing Workshop (HCW' 95)*, pp. 93-100, Apr. 1995.

[17] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, " Heterogeneous computing: challenges and opportunities," *IEEE Computer,* 26(6):18-27, June 1993.
[18] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environment," *4th Heterogeneous Computing Workshop (HCW' 95),* pp. 30-34, Apr. 1995.
[19] Legion Web Page. *http://legion.virginia.edu*.
[20] M. Maheswaran and H. J. Siegel, "A Dynamic matching and scheduling algorithm for heterogeneous computing systems," *7th Heterogeneous Computing Workshop (HCW '98),* pp. 57-69,March 1998.
[21] R. Moore, C. Baru, R. Marciano, A. Rajasekar, and M. Wan, "Data-intensive computing," in *The Grid: blueprint for new computing infrastructure,* I. Foster and C. Kesselman, ed., Morgan Kaufmann Publishers, San Francisco, CA, 1999, pp. 105-129.
[22] MSHN Web Page. *http://www.mshn.org*.
[23] B. Shirazi, M. Wang, and G. Pathak, "Analysis and evaluation of heuristic methods for static task scheduling," *Journal of Parallel and Distributed Computing,* 10:222-232, 1990.
[24] P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environment," *5th Heterogeneous Computing Workshop (HCW' 96),* pp. 98-117, Apr. 1996.
[25] G. C. Sih and E. A. Lee, "A Compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Trans.on Parallel and Distributed Systems,* 4(2):175-187, Feb. 1993.
[26] L. Smarr and C. E. Catlett, "Metacomputing," *Communications of the ACM,* 35(6):45-52, June 1994.
[27] Lee Wang, Howard Jay Siegel, Vwani P. Roychowdhury, and Anthony A. Maciejewski, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach," *5Journal of Parallel and Distributed Computing,* 47(1):8-22, Nov. 1997.

Biographies

Ammar Alhusaini is a Ph.D. candidate in the Department of Electrical Engineering - Systems at the University of Southern California, Los Angeles, California, USA. His main research interest is task scheduling in heterogeneous environments. Mr. Alhusaini received a B.S. degree in computer engineering from Kuwait University in 1993 and M.S. degree in computer engineering from the University of Southern California in 1996. Mr. Alhusaini is a member of IEEE, IEEE Computer Society, and ACM.

Viktor K. Prasanna (V.K. Prasanna Kumar) is a Professor in the Department of Electrical Engineering - Systems, University of Southern California, Los Angeles. He received his B.S. in Electronics Engineering from the Bangalore University and his M.S. from the School of Automation, Indian Institute of Science. He obtained his Ph.D. in Computer Science from Pennsylvania State University in 1983. His research interests include parallel computation, computer architecture, VLSI computations, and high performance computing for signal and image processing, and vision. Dr. Prasanna has published extensively and consulted for industries in the above areas. He is widely known for his pioneering work in reconfigurable architectures and for his contributions in high performance computing for signal and image processing and image understanding. He has served on the organizing committees of several international meetings in VLSI computations, parallel computation, and high performance computing. He also serves on the editorial boards of the Journal of Parallel and Distributed Computing and IEEE Transactions on Computers. He is the founding chair of the IEEE Computer Society Technical Committee on Parallel Processing. He is a Fellow of the IEEE.

Cauligi Raghavendra is a Senior Engineering Specialist in the Computer Science Research Department at the Aerospace Corporation. He received the Ph.D degree in Computer Science from University of California at Los Angeles in 1982. From September 1982 to December 1991 he was on the faculty of Electrical Engineering-Systems Department at University of Southern California, Los Angeles. From January 1992 to July 1997 he was the Boeing Centennial Chair Professor of Computer Engineering at the School of Electrical Engineering and Computer Science at the Washington State University in Pullman. He received the Presidential Young Investigator Award in 1985 and became an IEEE Fellow in 1997. He is a subject area editor for the Journal of Parallel and Distributed Computing, Editor-in-Chief for Special issues in a new journal called Cluster Computing, Baltzer Science Publishers, and is a program committee member for several networks related international conferences.