A Federated Model for Scheduling in Wide-Area Systems

Jon B. Weissman
Division of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249
(jon@ringer.cs.utsa.edu)

Andrew S. Grimshaw
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
(grimshaw@virginia.edu)

Abstract

In this paper a model for scheduling in wide-area systems is described. The model is federated and utilizes a collection of local site schedulers that control the use of their resources. The wide-area scheduler consults the local site schedulers to obtain candidate machine schedules. A set of issues and challenges inherent to wide-area scheduling are also described and the proposed model is shown to address many of these problems. A distributed algorithm for wide-area scheduling is presented and relies upon information made available about the resource needs of user jobs. The wide-area scheduler will be implemented in Legion, a wide-area computing system developed at the University of Virginia.

1.0 Introduction

Over the past few years the feasibility of wide-area high-speed network technology has been demonstrated by the implementation of network testbeds including Casa, ESnet, vBNS, etc. In 1995, an operational wide-area high-speed network known as I-way connecting machines at nineteen nationwide sites including Argonne National Laboratory in Illinois, the Cornell Theory Center in New York, and California Institute of Technology was demonstrated at Supercomputing in San Diego and deemed to be highly successful. Demands for network bandwidth on a national scale as evidenced by the explosion of internet traffic suggest that testbeds such as the I-Way will become more commonplace.

Traditionally, the guiding principle for distributed systems organization has been geographic proximity. The advent of gigabit networks offers another possibility. If the communication capacity between wide-area sites is sufficiently large then using remote resources for high performance computing becomes feasible. When the inter-site communication capacity is within an order of magnitude of the intra-site communication capacity, then it becomes beneficial to consider wide-area sites. For this reason, a model of distributed systems organization based on network capacity rather than geographic proximity is proposed.

The presence of these networks has created a paradigm for computing called *wide-area computing*. Wide-area computing is not entirely new. Loosely-coupled wide-area applications such as E-mail have been with us for some time. What is new is the possibility of obtaining high performance for more tightly-coupled applications or jobs using wide-area resources. There are two opportunities for achieving high performance in wide-area systems, reduced completion time for jobs, and high job throughput. For a single job, high performance can be obtained by either utilizing the resources contained in *multiple* sites or choosing the best *single* site. Essential to the success of wide-area computing is scheduling. The general scheduling problem is NP-complete [13] and numerous heuristics have been developed. Scheduling is the process of deciding where a job and its constituent tasks will run. A *schedule* is the set of machines assigned to a job. This paper describes the issues and constraints inherent in wide-area scheduling, and proposes the design of a wide-area scheduler for the Legion parallel processing system [8].

A number of research projects have addressed infrastructure support for wide-area computing. Infrastructure support includes remote execution, managing heterogeneity, shared file systems, etc. Two of these projects are Schooner [9] and Legion [8]. Perhaps the most popular wide-area system is the World-Wide Web that provides support for data sharing, but is not geared to high performance computing. The most widely used local-area distributed system is PVM [12], but it is limited to support for communication, heterogeneity, and remote execution.

Other projects deal specifically with achieving high performance via scheduling, but do so in a restricted distributed system such as a *NOW* or *metasystem.* Metasystem projects include [2][4][5]. NOW projects include Condor [10], DQS [11], and LSF [18]. All of these efforts concentrate on local as opposed to wide-area scheduling. Much of the NOW scheduling research is based on classic distributed system scheduling models [1][3].

Legion is a project designed to provide a comprehensive wide-area computing infrastructure with support for remote execution, shared file systems, fault tolerance, security, and high performance. Legion and its predecessor Mentat [7] have addressed the scheduling problem in a local-area context [6][14][15] and the wide-area scheduler described here will be implemented in Legion.

The remainder of this paper is as follows. Section 2.0 describes a set of objectives and issues that must be addressed in the design of a wide-area scheduler. Section 3.0 proposes a system model for user jobs that details what job information is needed for scheduling, and a network model for organizing the wide-area resources based on sites. Section 4.0 presents the wide-area scheduling algorithm and its implementation. Section 5.0 is the conclusion and future work.

2.0 Objectives and issues

Before embarking on the components of the design, it is important to detail the objectives that a wide-area scheduler must meet. The objectives of a wide-area scheduling system are the following:

- *Provide high performance for jobs*

  High performance is defined in the following simple way. If a performance benefit can be obtained using remote (i.e., wide-area) resources, then the scheduling system should use them. Otherwise, local resources should be used. The system may opt to use resources in the "best" single remote site or multiple sites.

- *Exploit resources in remote sites*

  This follows from the high performance principle. When appropriate, the wide-area scheduler should exploit remote resources.

- *Seamless to the user*

  The complexity of wide-area scheduling should be hidden from the user as much as possible. However, the user will be required to provide information about their job.

- *Maintain local autonomy for scheduling decisions*

  This is a critical design objective. The wide-area scheduler must make decisions in concert with the local site schedulers. The local site schedulers have authority over their resources and make scheduling decisions based on local needs. This is the basis of the *federated* scheduling model. For example, the local scheduler may use priorities, restrict parallel jobs to certain machines, etc. Methods for local job scheduling are discussed in [1][3][17]. The wide-area scheduler simply consults the local site scheduler to find out the best possible schedule that is available for a particular job.

- *Scalable run-time scheduling*

  The unpredictable pattern of resource sharing in a large, distributed, wide-area system means that scheduling must be deferred until run-time. Consequently, scheduling must be reasonably efficient. Furthermore, scheduling must be scalable in the sense that as more sites are added to the system, the scheduling performance degrades gracefully.

- *Handle scheduling constraints*

  Certain classes of jobs will have constraints that the scheduling system must accommodate. Three examples of constraints are the following. (1) a job must run in a particular site for reasons such as security or to be near data sources, (2) a job may require specific resource types, and (3) a job must run in under *n* time units. In the absence of constraints, the system will schedule the job to complete as quickly as possible.

Achieving these objectives requires that several issues and obstacles be addressed:

- *Data locality*

  It is assumed that part of the system infrastructure is the ability to access data and files independent of location, i.e., a shared file system is available. For jobs that are scheduled in a site that does not have direct access to needed data, the data must be transported to job's location. The scheduling system assumes that the data transport cost can be amortized over the course of the job execution. If this is not the case, then the job must be moved to the data and this must be expressed as a scheduling constraint.

- *Site selection*

  In a large wide-area system composed of hundreds to thousands of sites, it will not possible to consider all sites for a scheduling decision. A mechanism for identifying an appropriate subset of sites for scheduling is an important part of a wide-area scheduler.

- *Fault tolerance*

  A wide-area scheduler must be fault tolerant since the mean-time-to-failure for any component in a large, distributed system is very small. This also applies for jobs that are more vulnerable to faults due to wide-area distribution. Fault tolerance can be supported by a range of standard techniques including replication, elections, checkpoint-restore, etc. Full treatment of this problem is outside the scope of this paper.

- *Dynamic load balancing*

  The wide-area scheduler makes a run-time scheduling decision based on consultation with the local site schedulers. A decision to dynamically rebalance a job or set of jobs due to resource sharing, faults, or the irregular nature of a job, is a *local* decision. If the dynamic load balancing occurs locally, then the wide-area scheduler does not participate. However, the local site scheduler may issue a request to the wide-area scheduler that a job be re-scheduled elsewhere. Issues relating to *how* to dynamically load balance a running job are outside the scope this paper. All that can be said is that the wide-area scheduling system must not preclude this capability.

A number of system infrastructure issues must be solved in order for wide-area scheduling to be feasible, but are outside the scope of scheduling per se. These include:

- *Shared file system*
- *Naming and communication support*
- *Location and generation of task binaries for jobs*
- *Support for remote execution*
- *Management of heterogeneity*

Legion currently provides solutions for restricted cases of these general problems. In particular, near full support for networks of Unix workstations is provided. Support for a wider range of architectures and networks is planned.

For the wide-area scheduler to be effective, information about job resource needs must be available to evaluate possible scheduling decisions and to exploit remote resources. The availability of such information is the cornerstone of the proposed approach. This information is provided by the job and made available to the scheduling system. The wide-area scheduler is based on this job model and a network model. These are described next.

3.0 System model

3.1 Network organization

The network is organized as a collection of *sites*. A site is an administrative domain with certain security policies, file systems, resource accounting procedures, and most importantly, computing resources. It is unimportant to the wide-area scheduling system how the sites themselves organize their resources. A model for site organization based on processor clusters is described in [14][16]. Each site runs a *scheduling manager* (*SM*) and a *local scheduler* (*LS*). The *SM*'s run the wide-area scheduling algorithm discussed later in Section 4.0. The *SM* contains two components. One component is an interface to the site *LS* and the other is the wide-area interface between other *SM*'s. The *SM*'s and *LS*'s are address-space disjoint and communicate via message-passing. However, in some systems, the *SM* and *LS* may be configured as a single process.

The *LS* is responsible for managing the local site resources in a manner transparent to the *SM*. It is the *LS* that decides what resources it will make available for wide-area scheduling at any point in time. The *LS* contains two components. One component is an interface to the *SM* and the other is an interface to a site-specific scheduling system (e.g., NQS, Condor, Mentat, etc.). In the absence of a site-specific scheduling system, the *LS* software will provide a default scheduling capability. As an example consider two sites as depicted in Figure 1; site 1 contains a single parallel supercomputer (perhaps an Intel Paragon or SP-2), and site 2 contains a *NOW*. The *LS* for site 1 uses

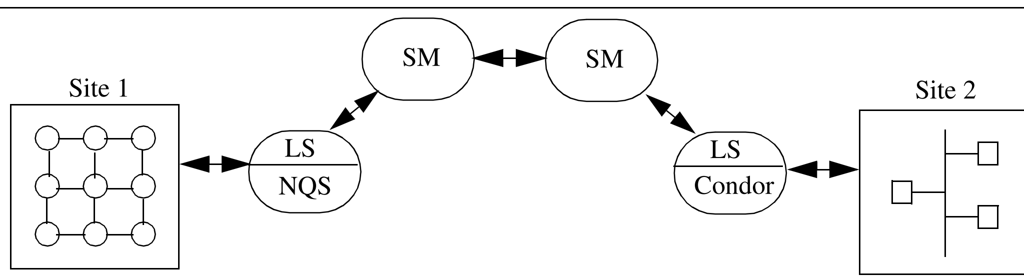

Figure 1: Example of federated scheduling

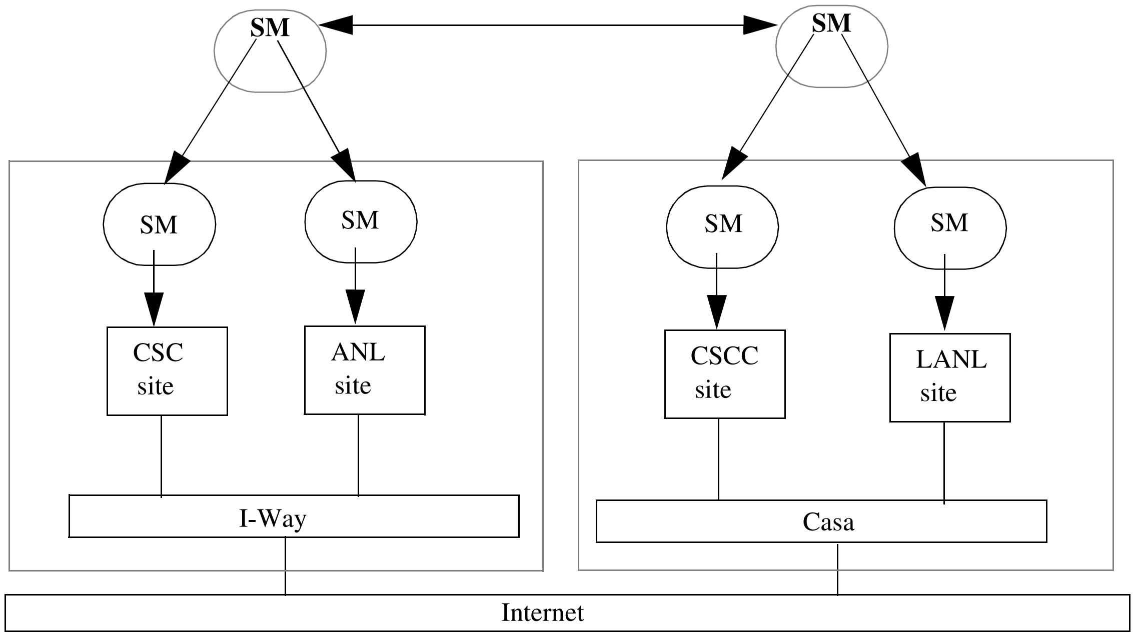

Figure 2: Virtual sites

NQS and the *LS* for site 2 uses Condor. NQS manages the supercomputer partitions and Condor manages the workstations. Constructing an *LS* will require modification to the site-specific scheduling systems to add additional interface functions. The changes will be confined to the scheduling daemons. For sites that contain a multitude of resources, several *LS*'s may be used with a single representative designated to communicate with the site *SM*. This is a configuration issue that does not concern the *SM*. The requirement of the *SM* interfaces are described shortly. Although shown as external to the site, the *SM* will be designated to run on one of host computers within a site.

For sites that have sufficient communication capacity between them, scheduling a single job across these sites may be feasible. This requires a tighter integration of the site *SM*'s through the *virtual site* mechanism. A virtual site is composed of multiple sites with sufficient communication capacity to be considered as a single site by scheduling. It has a single *SM* (shown in bold) that interacts with the constituent site *SM*'s instead of the *LS* (the site *LS*'s are not shown). As an example consider the I-Way and Casa virtual sites as depicted in Figure 2 (the virtual sites are the dotted boxes). A single job could be scheduled across the multiple sites connected on the I-Way given this organization.

3.2 Job model

Jobs may arrive at any computer in the system at any time. A job contains a set of concurrent tasks. A master-slave model is assumed in which the master task corresponds to the main program, see Figure 3. The master initiates the job scheduling request and waits for the result of job execution (if any). The bulk of the computation is carried out by the slaves. Sequential jobs contain only one slave task and data parallel jobs may contain a single SPMD slave task template that may be instantiated multiple times. For data parallel jobs, the degree of task instantiation depends on problem and resource characteristics [14]. The tasks of the job may communicate in some manner and this pattern may be known apriori. For task parallel jobs, the slave tasks may be executing different computations. The job implementation is assumed to have been produced by a programmer or compiler.

A job may be submitted to the wide-area scheduling system by sending it to the local *SM* which is located at a well-known local address. This is accomplished by the master. The implementation of this mechanism is outside the scope of this paper. It would be accomplished by a combination of language and library extensions as provided in Legion. Local jobs submitted to computers by other means are not scheduled by the wide-area scheduling system. However there is nothing that precludes these jobs from being scheduled by a local *LS*. From this point on,

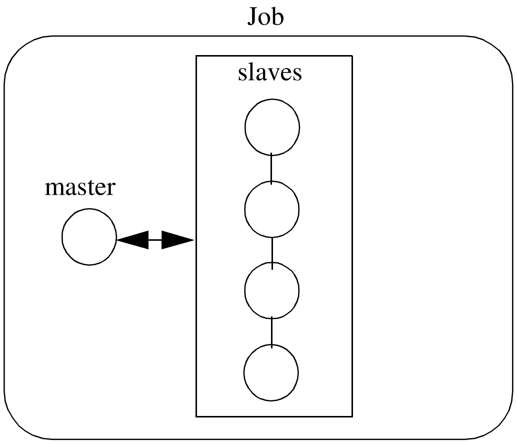

Figure 3: Master-slave job model

the term "jobs" refer to jobs submitted to *SM*'s only. The job mix is assumed to contain both sequential and parallel/distributed jobs. Jobs must provide information about their expected execution performance that can be exploited by the site *SM*. How this information will be used is deferred to the next section. This information allows the site *SM* to evaluate the effectiveness of candidate schedules for the job. The effectiveness is simply the predicted completion time for the job. How this information is made available is an implementation issue and outside the scope of this paper. A technique for encapsulating execution information for data parallel jobs using library functions called *callbacks* was implemented for the Mentat Programming Language (MPL) [14][15]. The following job information may be used:

- *Job type*

  Data parallel, task parallel, mixed-paradigm, distributed, or sequential. A distributed job is a collection of very loosely-coupled tasks that may be scheduled in multiple sites. If a parallel job is very loosely-coupled then it should be designated as distributed.

- *Job computation and communication*

  Task computation and communication costs must be provided. A job may have multiple implementations and a family of cost functions may be specified. In many cases this cost information may be specified abstractly in terms of instruction counts for computation and message size for communication. This is important because it must be possible to estimate these costs for heterogeneous machines. A model for data parallel programs demonstrated that this abstract information can be turned into very precise execution time estimates in a heterogeneous environment [16]. Some of the job information will require off-line benchmarking.

- *Scheduling histories*

  If a job has been run previously by the wide-area scheduling system, then the results of this previous schedule may be useful. The wide-area scheduling system will return a record of the scheduling decision and the elapsed time in some form for each scheduled job. Scheduling histories can be used for candidate site selection.

- *Scheduling constraints*

  These were discussed earlier. A language for specifying job scheduling constraints is an implementation issue. Related to constraints is the notion of preferences which are a weaker form of constraints. For example, a job may indicate site preferences, but the system is free to consider other sites as well.

- *Job-specific schedulers*

  The *SM* will contain a suite of scheduling algorithms for exploring and evaluating the set of candidate schedules by default. However, in some cases jobs may specify their own schedulers. For instance a data parallel job may indicate that it wishes to use the Prophet data parallel scheduler as developed for Mentat [14]. This capability should be supported.

4.0 Wide-area scheduling

4.1 The algorithm

The wide-area scheduling algorithm (WA) is a distributed algorithm that has two parts, the local site *SM* component, and the remote site *SM* component, see Figure 4. It is initiated by the arrival of a scheduling request to a local site *SM* from a master task running on a computer in the local site. When a job first arrives at a local site *SM*, a set of candidate sites are selected (*including* the local site) and the job is sent to each remote site *SM* for *bidding*. Selecting the candidate sites is discussed later. Each remote site *SM* then contacts its *LS* to get a list of machines and when they are available. Additional information may also be returned such as whether the machines are shared or may be dedicated to the job. The remote *SM* then runs a local scheduling algorithm to search this set of machines to evaluate the possible schedules taking job constraints into account. The best schedule and projected completion time are returned to the local site *SM*. The best schedule has the smallest predicted completion time. The local site *SM* then selects the best site, and initiates the scheduling of the job

1. local SM receives WA_sched (Job) request
2. local SM chooses k candidate site managers $SM_1 \ldots SM_k$
3. for each SM_i
    call $best_i = SM_i .get\_best$ (Job)
4. the_best = min $\{best_i. elapsed\_time\}$ over all i in $1 \ldots k$
5. for each SM_i
    if ($SM_i == the\_best.site$)
        $result = SM_i .go\_SM$ (Job, the_best)
    else
        $SM_i .no\_go\_SM$ (Job, the_best)
6. Return *result* and *scheduling info* to caller

a) Local SM component

1. remote SM receives *get_best* (*Job*) from local SM
2. call $scheds = LS.get\_scheds$ (Job)
3. call $best = eval\_scheds$ (scheds, Job)
4. lock *best,* return *best* to local SM
5. if remote SM receives *go_SM* (*Job, the_best*)
    $result = LS.go\_ls$ (Job, the_best)
    return *result* to local SM
  else if remote SM receives no*_go_SM* (*Job*, *the_best*)
    store *the_best* in table
6. release lock on *best*

b) Remote SM component

Figure 4: WA algorithm

on that site. This algorithm offers an opportunity to provide robust job scheduling – instead of a single site selection, the system could redundantly schedule the job in multiple sites for fault tolerance. This possibility is being investigated.

The remote *SM* locks its best schedule until the scheduling transaction is complete. Locking is used to prevent race conditions. If another scheduling transaction is initiated in the midst of the current one, there may be a race for the machines in candidate schedules of both transactions. To prevent this, the first transaction to initiate *eval_scheds*, locks its best schedule within the remote *SM*. Locking simply makes the machines contained in schedule unavailable for future transactions until they are released. When the local *SM* receives all[1] of the bids, it selects the best schedule, and informs all of the remote sites of its decision. This information can be used by the other sites in making future scheduling decisions. For example, the other sites can avoid this site in the short-run since it is already running a job. The sites can purge this information from their tables after *elapsed_time* time units (the projected completion time) have passed. The remote *SM* that was selected passes the job to the local *LS* for execution. At this point, the other remote *SM*'s can release the lock on their best candidate schedule. The procedure calls across address-spaces (e.g., *get_best*) are RPC's. The interactions between the various system components are depicted in Figure 5 (site SM_1 is selected for the job). The implementation of the WA algorithm requires that the *SM* and *LS* support several interface functions. These are described next.

4.2 The interfaces

SM ↔ SM

The following functions must be provided on the *SM* ↔ *SM* interface within the *SM*:

- *get_best* (*Job*) → *schedule* + *elapsed_time*

  Returns best schedule for the job and the projected elapsed time for its execution within the site managed by called *SM*. The schedule is a set of machines.

- *go_SM* (*Job, schedule*) → *result* + *scheduling info*

  Initiates job on called *SM* using the machines contained in schedule. It returns a result and scheduling information to the caller. The scheduling information includes the elapsed time taken by the job.

- *no_go_SM* (*Job, schedule*) → *void*

  Informs called *SM* that job will be scheduled elsewhere (using schedule). This serves as an acknowledgment to close the transaction. It also provides future scheduling information to the called site. This is a void function so the caller need not wait for its completion.

1. Performance may dictate a time-out period for replies and perhaps only a subset of the replies will be waited for.

SM ↔ LS

The following functions must be provided on the *SM* ↔ *LS* interface within the *LS*:

- *go_LS* (*Job, schedule*) → *result + scheduling info*

  Initiates job on called *LS* using the machines contained in schedule. It returns a result and scheduling information to the caller. The scheduling information includes the elapsed time taken by the job.

- *get_scheds* (*Job*) → *schedules + availability +shared + cost*

  Returns the set of possible machine schedules, when they are available, whether the machine is shared, and the cost of using a given machine(s). The *LS* decides which resources it will make available based on local system needs. A site *LS* may also provide mechanisms for controlling *when* machines are available. For example, Legion supports a machine reservation capability. If the machine is shared, then its current load status is also provided. A shared machine cannot give a performance guarantee to the job. For example, a multicomputer partition may be allocated in a dedicated fashion, but most workstations are shared. For a job that has a performance constraint, avoiding shared machines may be necessary. Cost is an abstract quantity which can be used by a site to charge for the use of its resources. For example, Legion generates resource accounting information for use in a cost policy. Note: *get_scheds* may require modification of existing *LS* software to make all of this information available.

Local Computer ↔ SM

The following functions must be provided on the *Local Computer* ↔ *SM* interface within the *LS*:

- *WA_sched* (*Job*) → *result* + *scheduling info*

  Send job scheduling request to local site *SM* which results in the job being scheduled in the best candidate site. Recall that this is sent from the master task component of the job (i.e., the main program). If the job returns a result it is passed back to the master task that initiated the *WA_sched* call. The scheduling information includes the selected schedule and elapsed time. It may be used for future scheduling decisions.

4.3 Inside the SM

In supporting the WA algorithm, a number of other issues arise. In step 2 of the WA scheduling algorithm, the site *SM* must select *k* candidate sites for scheduling. How is this done? First, each *SM* must know the addresses of the other site *SM's* in the system. These must be well-known addresses established at configuration time. Other configuration issues include where the *SM* and *LS* will run within the site. It is assumed that a particular site will only know the addresses of a subset of the sites in the entire system. For example, some sites will permit only a few, perhaps trusted sites to use their resources. Other sites will have an organizational connection such as NSF supercomputer sites or NASA sites. The total number of remote sites that a particular site may utilize for scheduling is *N*. However at any point in time, the number of *active* remote

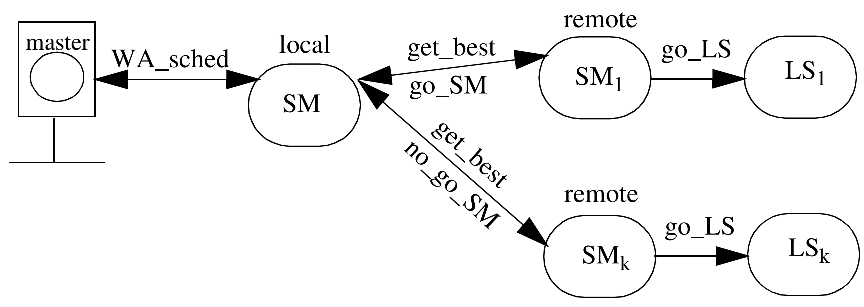

Figure 5: WA coordination

sites is M, $M \leq N$. An inactive site will not used for scheduling. A site may become inactive by any number of means. A remote site may tell this site to stop sending jobs for a certain time period. Another example is that the remote site has suffered some type of fault or is otherwise inaccessible. Temporal information may also be useful. For example, certain times of the day are busier and this information can be used. The *SM* will keep a table of the active sites. The *SM* will also keep a table of the jobs and their location (sites) that have been scheduled on its behalf. Below the *SM* functions are described and some open issues.

- *WA_sched* (*Job*) $\rightarrow$ *result* + *scheduling info*

  This is executed by the local site *SM* when the main program of a user job requests wide-area scheduling. The details of this function are described above and in Figure 4. An open issue is how to decide on the k candidate sites for scheduling ($k \leq M$). The problem is simplified if the job specifies constraints or preferences, or contains a history of good quality prior scheduling decisions. The number of sites selected should be related to the computational weight of the job. A long-running job can tolerate the additional overhead associated with the consideration of a larger number of candidate sites. A metric that relates the computation weight to k needs to be developed. A secondary issue is which subset of k sites ought to be selected? If the local site has no knowledge of the resources and their availability in the other sites, then a random selection may work well. The local site should probably avoid sites that are already executing jobs on its behalf.

- *eval_sched* (*scheds*, *Job*) $\rightarrow$ *best schedule* + *predicted elapsed time*

  The local *SM* selects the best candidate schedule from the set *scheds* by invoking this function. The best schedule is based on local system needs. For example, some site *SM*'s may be configured for high throughput, bias to parallel or sequential jobs, reduced completion time, etc. The selected schedule together with an estimate of the job completion time using this schedule is returned. Estimating job completion time requires information about the job implementation described earlier. A model for accurately predicting the run-time for data parallel jobs in heterogeneous workstation networks is described in [14]. This model will be extended for other job classes. It is envisioned that a suite of evaluation algorithms will be needed for the other job classes.

5.0 Conclusion and future work

The objectives and issues that arise in the design of a wide-area scheduling system were described. A federated model for a wide-area scheduling system was presented that meets many of these objectives. The system architecture is scalable and preserves local scheduling autonomy. The local site decides which resources it will commit to a new job based on local system needs.

The system will schedule jobs remotely if there is a performance benefit in doing so. A distributed scheduling algorithm that selects the best candidate schedule from among a set of available sites was described. This algorithm relies on a local algorithm that is used by a site to evaluate the set of possible machine schedules. This evaluation procedure relies upon information made available about the user jobs. An evaluation procedure for data parallel jobs has been developed and will be extended for other job classes. The system must also support job preferences, scheduling constraints, and exploit job scheduling histories, if available. It must also allow jobs to specify their own schedulers if they wish. Future research will address several open issues including candidate site selection for jobs. A prototype implementation of the wide-area scheduler in the Legion system is planned to demonstrate the feasibility of the approach.

6.0 References

[1] T.L. Casavant, and J.G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," *IEEE Transactions on Software Engineering*, Vol. 14, February, 1988.

[2] H.G. Dietz, W.E. Cohen, and B.K. Grant, "Would you run it here... or there? AHS: Automatic heterogeneous supercomputing," *Proceedings of the 1993 International Conference on Parallel Processing*, 1993.

[3] D.L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Transactions on Software Engineering*, Vol. 12, May 1986.

[4] F. Freund and H.J. Siegel, "Heterogeneous Processing," *IEEE Computer*, June 1993.

[5] A.S. Grimshaw, J.B. Weissman, E.A. West, and E. Loyot, "Metasystems: An Approach Combining Parallel Processing And Heterogeneous Distributed Computing Systems," *Journal of Parallel and Distributed Computing,* Vol. 21(3), June 1994.

[6] A.S. Grimshaw and V. E. Vivas, "FALCON: A Distributed Scheduler for MIMD Architectures", *Proceedings of the Symposium on Experiences with Distributed and Multiprocessor Systems*, Atlanta, GA, 1991.

[7] A.S. Grimshaw, "Easy to Use Object-Oriented Parallel Programming with Mentat," *IEEE Computer*, May 1993.

[8] A.S. Grimshaw, A. Nguyen-Tuong, and W.A. Wulf," Campus-Wide Computing: Early Results Using Legion at the University of Virginia, CS-95-19, March 1995.

[9] P.T. Homer and R.D. Schlichting, "A Software Platform for Constructing Scientific Applications from Heterogeneous Resources," *Journal of Parallel and Distributed Computing,* Vol. 21(3), June 1994.

[10] M.J. Litzkow et al, "Condor - a hunter of idle workstations," In *Proceedings of the 8th International Conference on Distributed Computing Systems*, June 1988.

[11] L. Revor, *DQS Users Guide*, Computing and Telecommunications Division, Argonne National Laboratory, September 1992.

[12] V.S. Sunderam, "PVM: A framework for parallel distributed computing," *Concurrency: Practice and Experience*, Vol. 2(4), December, 1990.

[13] J. Ullman, "NP-complete scheduling problems," *Journal of Computing System Science*, Vol. 10, 1975.

[14] J.B. Weissman and A.S. Grimshaw, "A Framework for Partitioning Parallel Computations in Heterogeneous Environments," *Concurrency: Practice and Experience*, Vol. 7(5), August 1995.

[15] J.B. Weissman and A.S. Grimshaw, "Network Partitioning of Data Parallel Computations," *Proceedings of the Third International IEEE Symposium on High Performance Distributed Computing*, August 1994.

[16] J.B. Weissman, "Scheduling Parallel Computations in a Heterogeneous Environment," Ph.D. dissertation, University of Virginia, August 1995.

[17] J.B. Weissman, "The Interference Paradigm for Network Job Scheduling," *Heterogeneous Computing Workshop, 10th International Parallel Processing Symposium IPPS*, April 1996.

[18] S. Zhou et al, "Utopia: A Load Sharing Facility for Large Heterogeneous Distributed Computer Systems," *Software: Practice and Experience*, Vol. 23(12), December 1993.