Network Centered Multiple Resource Scheduling in e-Science Applications

Yan Li, Sanjay Ranka, Sartaj Sahni, and Mark Schmalz

Department of Computer and Information Science and Engineering
University of Florida, Gainesville, Florida 32611
{yanli,ranka,sahni,mssz}@cise.ufl.edu

Abstract. Workflows generated from e-Science applications require the simultaneous use of multiple types of resources. In this paper, we present the Multiple Resource Reservation Model ($MRRM$), which enables the monitoring and scheduling of multiple heterogeneous and distributed resources. $MRRM$ provides a unified representation for multiple types of distributed resources, and represents resource constrains such as compatibility and accessibility. Using $MRRM$, we solve the Multiple Resource First Slot problem based on a collection of algorithms that are customized for different request and resource types. Our simulation results demonstrate efficiency and scalability of our algorithms for various sizes of networks and resource sets.

1 Introduction

Many e-Science applications require the simultaneous use of multiple types of resources. To fulfill such multiple resources requirements, we propose a framework for conducting admission control and in-advance reservations on various computational resources such as network bandwidth, CPU, storage and even software licenses.

Resource Reservation has been long studied as an important approach to provide QoS guarantee. In the network field, bandwidth reservation is the major concern. Either **on-demand scheduling** or **in-advance scheduling** has been characterized by a set of algorithms and systems [1, 2, 3, 4, 5]. On the other hand, in system and architecture field, many paper also focused on CPU of storage resources reservation. [6, 7, 8]. For example, Maui [8], which is a popular high-performance computing scheduler for clusters and supercomputers, do not schedule network resources, although they are able to do advance reservation of resources such as CPU and storage. On the other hand, network bandwidth management systems such as those for UltraScienceNet (USN) and ESnet do not schedule computer resources. The Sharc system [7] modeled both network and CPU resources in the clusters as unified resources blocks, but the constrains among resources, such as network topology, resource compatibilities are not considered.

In this paper, we envisioned those large scale e-Science applications which consist of dedicated networks with hundreds of nodes, and computational resources

T. Doulamis et al.: (Eds.): GridNets 2009, LNICST 25, pp. 37–44, 2010.

with different platforms. When multiple resources are reserved, their topologies, dependencies, and compatibilities must be handled. The purpose of our paper is to develop a co-scheduler that simultaneously schedules multiple types of resources with a network focus based on a our Multiple Resources Reservation Model ($MRRM$). We also solved a Multiple Resource First Slot problem ($MRFS$) with the objective of determining the earliest time that can be used to reserve all resources required to compute a given request.

The rest of the paper is organized as follows. In Section 2 we give the detail of our resource model $MRRM$ and define the related data structures. In Section 3 we introduce the $MRFS$ problem and formally define the request pattern. In Section 4, we propose our algorithms for multi-resource scheduling. In Section 5, we evaluate these algorithms to show their effectiveness. The conclusions are drawn in Section 6.

2 Resource Model and Data Structure

2.1 Resource Model: $MRRM$

In this paper, the Multiple Resources Reservation Model ($MRRM$) is defined to provide a unified presentation for different types of resources. In e-Science applications, resources can be classified into network resources and local computational resources. Network resources, including links, routers and switches, transfer user's data from one site to another. Local computational resources include CPU, storage and other resources that are used in processing user requests. $MRRM$ is modeled as a graph $G < V, E >$ with the communication network in the middle, and computational resources attached to network edge nodes. Each switch or router is represented as a node in V, while each network link is mapped to an edge in E. The local computational resources attached to one of the edge routers. A single resource unit is modeled as an edge in that subgraph.

As the resources maybe of different types, we assign each resource link a type ID ($T-ID$), to specify its type - for example, CPU, Memory. With the type ID, all local resources can be grouped into one of several multi-partitioned resource constraint graphs (MPRCGs), which is presented in Figure 1. Three resource partitions are presented:Resource A, B and C. However, $T-ID$ is not enough, as the resources with same type(CPU) may not be cooperate together(X86 and MIPS). So we also provide a compatibility ID (C-ID) to each resource. $C-ID$ facilitates grouping of resources with the same T-ID into smaller groups of compatibility, as shown in the smaller grey cycles in partition A and C in Figure 1. With both $T-ID$ and $C-ID$, user's preference on various resources can be explicitly specified.

A further consideration involves accessibility. For example, some computers use Distributed Shared Memory to provide all CPU nodes full access to the memory model. However, other systems only allow certain CPUs to access specific memory partitions. Our MRRM is capable of handling both scenarios. If two resource links within different MPRCGs are accessible to each other, then a specific auxiliary link with unlimited capacity connects the two links. The connections

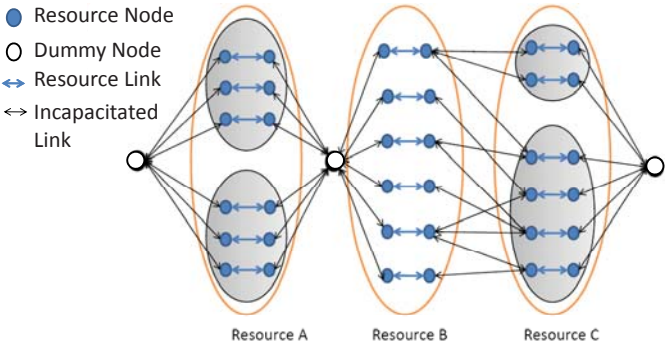


Fig. 1. Detailed Model of MRRM

between partition B and C in Figure 1 give an example of such scenarios. If all resources in one MPRCG are accessible to all resources in another MPRCG, then a connection is made at the MPRCG level, so that every resource link in either MPRCG is connected with every resource link in the other MPRCG, as shown in Figure 1 between partition A and B.

2.2 Data Structures

In $MRRM$, the status of each resource unit is maintained using a Time-Resource list (TR list). It is comprised of tuples of the form (t_i, r_i), where t_i is a time and r_i is the amount of resources. The tuples on a TR list are in increasing order of t_i. If (t_i, r_i) is a tuple of $TR[l]$ (other than the last one), then the amount of available resource l from t_i to t_{i+1} is r_i. When (t_i, r_i) is the last tuple, there are r_i units of resource l available from t_i to ∞.

We also employ two other data structures: ST(Start Time) List and Basic Interval. ST List, like TR List, is also associated with a resource link and is directed computed from the corresponding TR list and current request. First, we define, for each edge (u, v), an ST list $ST(u, v)$, which is comprised of pairs of the form (a, b), $a \leq b$. The pair (a, b) has the below interpretation: for every start time $x \in [a, b]$, edge (u, v) has enough resources from time x to time $x + dur$ to fulfill the current request. As proved in [5], The earliest start time is the smallest a_i for which there is an path p from s to d in the graph and every edge in p has a ST interval contains this a_i.

Basic Interval is a time interval $[t_i, t_{i+1}]$, where $t_i < t_{i+1}$ and within a Basic Interval, for any edge $e \in E$ in our multi-resource graph, its status, like bandwidth or disk spaces, remains static. Also, any two disjoint steady stages are independent from each other: any change on the link status in one Basic Interval will not affect the scheduling result in any other steady stages. To build a list that contain all the Basic Intervals, we first consider the time part of a edge's TR list, $t_0, t_1, t_2 ... t_q$, any time interval $[t_i, t_{i+1}]$ forms a steady stage of that link. Hence, if we union the time parts of each resource link's TR list together, we will obtain a list for all Basic Intervals.

3 Problem Definition

In **MRFS** scenario, the user's request to have the job starts as early as possible within a specific time window. If the request is feasible at some time within the reservation window, the earliest time t will be reported to the user and the corresponding resources will be reserved, or else, the request will be rejected.

A $MRFS$ request is a 6-tuple (s, d, dur, $ResWin < st, et >$, $RV < R_0, R_1, R_2... >$, $shareable$). s and d are the source and destination node of the data transfer, the computational resources attached to s and d are the computational resources that are going to be reserved. dur is the duration that those resources needs to be reserved. $ResWin < st, et >$ is the reservation window, which means the job's start time must be in the time interval $[st, et - dur]$. RV is a vector that contains the all resource requirements. $shareable$ flag specifies whether the job can be split among different resource units.

4 Multiple Resource Scheduling Algorithm

Based on whether job can be split and whether the local resources are fully accessible(i.e. resources in one layer can access part of the resources in its neighborhood layer or all of them), we divided the MRFS problem into 4 sub-problems:

WS-RC: The workload can be split, and local computational resources are constrained on accessibility.

WN-RC: The workload cannot be split, but local computational resources are constrained on accessibility.

WS-RN: The workload can be split, and local computational resources are not constrained on accessibility.

WN-RN: The workload cannot be split, but local computational resources are not constrained on accessibility.

4.1 $WS - RC$ Scheduling Algorithm

If the workload can be split, then given request r_i and the multi-resource graph G, our $WS - RC$ algorithm discovers the maximum flow from source's sink to destination's sink for each basic interval within the reservation window. The scheduler then attempts to identify (a) if there are enough resources within the current basic interval BI_i, if true, BI_i is marked as feasible; and (b) whether or not there exists one consecutive (i.e., temporally connected) sequences of basic intervals [BI_i, BI_{i+1}, ..., BI_j] with total length longer than the required duration dur. If such a sequence is found, then the earliest possible start time of the sequence becomes the first possible start time of the user's request. The detail of $WS - RC$ Scheduling Algorithm is shown in the left part of Figure 2.

To apply the traditional max-flow algorithm to the various times of resources, we need the scale the capacity of different types of resource links to one Base

```
WS-RC Scheduling (req_i, G) {
  Randomly choose resource r_k as base resource;
  G' = Scale(G, r_k);
  Build the global time list L from G', remove all the T_i
      outside the scheduling window req_i.ResWin;
  Identify all the Steady Stages;
  For each Steady Stage SS_i {
    MF_i = MaxFlow(G', SSi);
   if(MF_i ≥ basevalue)
     Mark SS_i as a feasible Steady Stage;
  }
  Traverse the Steady Stage list again. find out the
      first consecutive feasible steady stages list which
      is longer than req_i.dur;
  if (such list exist);
    accept the request;
    set list's the start time as request's start time;
  else
     reject the request;
}
```

```
Extended Bellman-Ford (s,d) {
  initialize st(∗) = st(0, ∗);
  // compute st(∗) = st(n − 1, ∗)
 put the source vertex into list1;
 for (int k = 1; k < n; k++) {
    // see if there are vertices
      whose st value has changed
    if (list1 is empty)
        break; // no such vertex
    while (list1 is not empty) {
        delete a vertex v from list1;
    for (each edge (v, u)) {
      st(u) = st(u) ∪ {st(v) ∩ ST (v, u)};
      if (st(u) has changed and
      u is not on list2)
          add u to list2;
    }
    list1 = list2; make list2 empty;
  }}}
```

Fig. 2. Algorithms for $WS-RC$(left) and $WN-RC$(right)

Value or the flow that is directly computed will not make any sense. Here, we chose network bandwidth as our base resource. If bandwidth's request be $10(MB/s)$ and CPU's request be $5(GHz)$, we scale all the CPU resource links' capacity by a factor of 2 and we also need to set the CPU to requirement from $5GHz$ to $10GHz$. In this way, we achieve the numeric unification among different type of resources in $MRRM$, which enables the min-cut algorithm [9] to be applied directly on our scaled graph.

The complexity of the above $WS-RC$ scheduling algorithm is $O(|SS_{RW}| * N^3)$, where $|SS_{RW}|$ is the size of Basic Interval list what is within the reservation window and $N = N_N + N_s + N_d$. N_N is the number of nodes in the network. N_s and N_d the number of local computational resources attached to s and d.

4.2 $WN-RC$ Scheduling Algorithm

In the $WN-RC$ case, the workload can only be transferred on a single path in the network and can only be processed using a single unit of each type local computational resources. So we use the Extended Bellman-Ford algorithm [5] to solve this problem. The algorithm is presented in the right part of Figure 2.

First, we will extend the concept of an ST list for an edge a path. Let $st(k, u)$ be the union of the ST lists for all paths from vertex s to vertex u that have at most k edges on them. Clearly, $st(0, u) = \emptyset$ for $u \neq s$ and we assume $st(0, s) =$

$[0, \infty]$. Also, $st(1, u) = ST(s, u)$ for $u \neq s$ and $st(1, s) = st(0, s)$. For $k > 1$ (actually also for $k = 1$), we obtain the following recurrence

$$st(k, u) = st(k-1, u) \cup \{\cup_{v:(v,u)} \text{ is an edge } \{st(k-1, v) \cap ST(v, u)\}\} \quad (1)$$

where $\cup$ and $\cap$ are list union and intersection operations. For an n-vertex graph, $st(n-1, d)$ gives the start times of all feasible paths from s to d. The Bellman-Ford algorithm [10] may be extended to compute $st(n-1, d)$.

It is easy to see that the computation of the $st(*, *)$s may be done in place (i.e., $st(k, u)$ overwriting $st(k-1, u)$) and the computation of the sts terminated when $st(k-1, u) = st(k, u)$ for all u. With the above observation, here we give the detail of Extended Bellman-Ford algorithm.

Each iteration of the **for** loop takes $O(L)$ time, where L is the length of the longest st list. Since this **for** loop is iterated a total of $O(N * E)$ times, the complexity of the extended Bellman-Ford algorithm is $O(N * E * L)$, where N and E is the number of nodes and links in the whole multiple resource graph.

When using the extended Bellman-Ford algorithm to solve the first slot problem, we first find the earliest start time t for a feasible path using Extended Bellman Ford algorithm. Then, the actual path may be computed using BFS where the feasibility of each link is computed by fixed the job's $t_{start} = t$ and $t_{end} = t + dur$. Also BFS guaranteed to find the shortest feasible path in the graph.

4.3 $WS - RN$ Scheduling Algorithm

In the $WS - RN$ scenario, although the $WS - RC$ scheduling algorithm can directly be applied, there still exists a much fast algorithm: First, we compute the available ST List in network from the source edge router to the destination edge router, using the $WS - RC$ algorithm. Second, we compute the available time slots for local resources by checking, for each computational resource partition at source and destination, whether any set of compatible resources can provide sufficient resources to satisfy the user's job request. Given a certain partition in a MPRCG, we group all the compatible resources together, and compute the aggregate TR list for this compatibility group. Then, we compute the ST list according to current request and union all the group ST List to produce a ST list for the current partitions. Finally, we intersect the network's ST list with the ST lists specific to all local resource partitions, to determine availability of a start time.

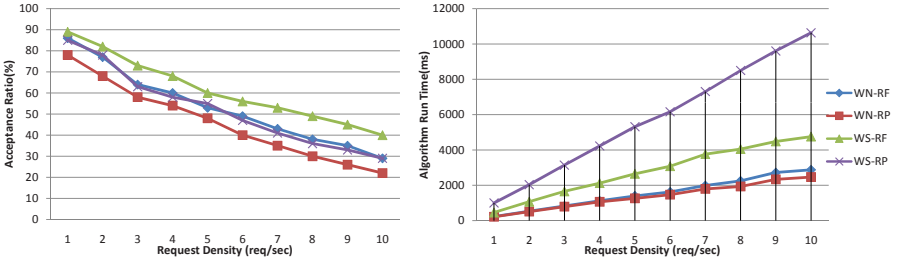
In this algorithm, we only check each local resource stage for each corresponding request by visiting each resource link for $O(1)$ times. Since the list union and intersection can also be finished in linear time, we can reduce the algorithm run time to $O(|SS_{RW}| * (N_s + {N_N}^3 + N_d)$.

4.4 $WN-RN$ Scheduling Algorithm

The $WN-RN$ problem is similar to the previous $WS-RN$ problem. In particular, a network path can be computed via the *Extended Bellman-Ford* algorithm to yield the first start time. This computation is followed by *breadth-first search* to identify the path. For local computational resources, we can apply the same approach as in $WS-RN$. However, $WN-RN$ neglects the grouping of resources according to compatibility. Since the requested job cannot be split, only one resource unit in each resource partition is required. In that case, the algorithm's complexity is bounded by $O(N_N * E_N * L + N_s + N_d)$

5 Evaluation

We tested our work on a USNET simulator at Oak Ridge National Laboratory (ORNL). The evaluation results are presented below.



(a) AR vs Density (b) AR vs Network Size

Fig. 3. Acceptance ratio changing with (a)request density; (b)network size.

Figures 3a Figures 3b provides the average acceptance ratio and algorithm run time as a function of request density, respectively. The result is acquired using a 100 node random network with 15 randomly selected local resources sites.

Our experiment results show the following:

1. The increase of request density will degrade every algorithms' performance. As more request come into the system within the same time interval, the network and clusters becomes congested, hence more requests were rejected since not enough resources are available. In the mean time, as more requests are running in the system simultaneously, the length of Basic Interval list and ST list increases, which leads to longer algorithms' run time.
2. When multiple paths are allowed and resources are fully shared, the scheduler can better utilize system resources, so as to accept more requests. However, the resulting multi-path algorithms require more computation time to obtain a feasible result.
3. Generally speaking, all 4 algorithms scales very well with either system size or request density. Even when the workload is high, one request averagely takes less than a minute to find out scheduling result.

6 Conclusion

In this paper, we consider the multiple resource scheduling problem, and present several solutions in terms of a multi-resource model. In e-Science, computational resources are widely distributed and are connected by fast optical networks. Thus, we propose a co-scheduling solution comprised of a flexible and efficient multi-resource reservation model ($MRRM$) and solve four instances of the multiple reservation first slot ($MFRS$) problem. Based on our model, four algorithms were developed to solve salient instances of $MRFS$. Experiments on a heterogeneous computer network showed that our algorithms are scalable linearly in terms of network size and request ratio.

Acknowledgement

This work was supported, in part, by the National Science Foundation under grant 0312038 and 0622423. Any findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF. The work was also supported in part by a grant from UltraHiNet and Florida High Tech Corridor.

References

1. Aukia, P., Kodialam, M., Koppol, P.V.N., Lakshman, T.V., Sarin, H., Suter, B.: RATES: A server for MPLS traffic engineering. IEEE Network, 34–41 (March/April 2000)
2. Guerin, R., Orda, A., Williams, D.: Qos routing mechanisms and ospf extensions. IETF Internet Draft (1996)
3. On-demand secure circuits and advance reservation system, `http://www.es.net/oscars`
4. Guerin, R.A., Orda, A.: Qos routing in networks witrh inaccurate information: Theory and algorithms. IEEE/ACM Transactions on Networking 7(3), 350–364 (1999)
5. Sahni, S., Rao, N., Ranka, S., Li, Y., Jung, E.S., Kamath, N.: Bandwidth scheduling and path computation algorithms for connection-oriented networks. In: Sixth International Conference on Networking (ICN 2007), p. 47 (2007)
6. Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., Tantawi, A.: An analytical model for multi-tier internet services and its applications. In: Proc. of ACM SIGMETRICS, pp. 291–302 (2005)
7. Urgaonkar, B., Shenoy, P.: Sharc: Managing cpu and network bandwidth in shared clusters. Technical report, IEEE Transactions on Parallel and Distributed Systems (2001)
8. Maui, `http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php/`
9. Ahuja, R., Magnanti, T., Orin, J.: Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Englewood Cliffs (1993)
10. Sahni, S.: Data structures, algorithms, and applications in C++, 2nd edn. Silicon Press (2005)