Scheduling Multi-Component Applications in Heterogeneous Wide-area Networks

Jon B. Weissman[1]

Department of Computer Science and Engineering
University of Minnesota
Minneapolis, MN 55455
651-291-8898
(jon@cs.umn.edu)

Abstract

In this paper we present a scalable scheduling heuristic for several common classes of multi-component applications (meta-applications). We consider this scheduling problem in a wide-area heterogeneous computing environment, or metasystem. The heterogeneity and scale of the computing environment and the heterogeneity of the application make this a challenging problem. We have studied the performance of the heuristic in simulation and the results are encouraging. Completion times for three common classes of meta-applications were within 10-20% of optimal on average with a worst-case variance of 60%. The results suggest that effective scheduling of meta-applications is possible, if sufficient application and system resource cost information is provided.

1.0 Introduction

Metacomputing is the seamless application of wide-area distributed computing resources to user applications. A number of research groups are building low-level metacomputing infrastructure [2][4]. What distinguishes metacomputing applications from other wide-area distributed applications is the objective of high-performance. In fact, it is the promise of performance greater than can be achieved using single site resources that makes metacomputing most attractive in solving complex scientific problems. The applications most suitable for metacomputing are often very heterogeneous in structure [6]. For instance, such applications may include remote databases or servers, remote instruments, remote supercomputers, VR devices, and humans-in-the-loop. In addition to hardware heterogeneity, the underlying networks that connect the different sites may also exhibit performance heterogeneity in both latency and bandwidth. The challenges inherent in heterogeneous computing are well known [3].

Exploiting the performance potential in metacomputing environments requires effective application scheduling: the selection and allocation of resources to the application. This problem is particularly challenging due to the heterogeneous nature of both the resources (machines and networks) and the application itself. In this paper, we consider the scheduling of *meta-applica-*

1. This work was partially funded by NSF ACR-9996418

tions; applications consisting of multiple schedulable components across multiple sites with the goal of reduced completion time. Previous work showed that selecting the best single site for single component parallel applications can be done efficiently [8]. The scheduling of multiple interacting components across multiple sites is a complex problem especially when the network capacity is assumed to vary between sites. Most of the metacomputing scheduling work assumes either communication between components can be ignored, or the application will be confined to run in a single site, or the number of sites and components are small enough to make a brute-force scheduling algorithm feasible. In contrast, we present a scalable scheduling heuristic that has achieved excellent results (typically within 10% of optimal on average) in a detailed simulation study of three common classes of meta-applications.

2.0 Meta-Application Model

Meta-applications consist of a set of distinct application *components* that may communicate and interact over the course of the application. Components may be schedulable computations, remote servers or databases, remote instruments, humans-in-the-loop, etc. (Figure 1). Computational components themselves may be sequential or parallel computations. Some components are fixed such as a remote database server and do not require scheduling, but may impact the scheduling of other components. For example, the placement of component B may be influenced by the amount of data transmitted by a database to B. If a great deal of data is moved, then a high-speed link between B and the database may be desired. It is also possible that other components are fixed due to scheduling constraints, such as a given program component must run in a particular site.

We consider meta-applications in which the inter-component communication pattern is statically known (Figure 1) and divide meta-applications into three categories — *concurrent*, *parallel*,

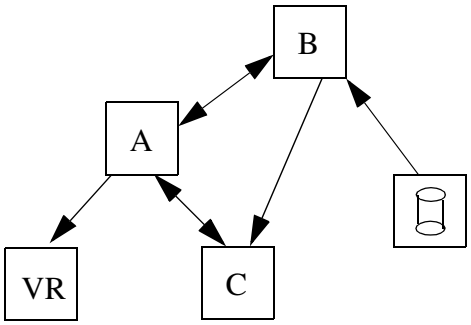
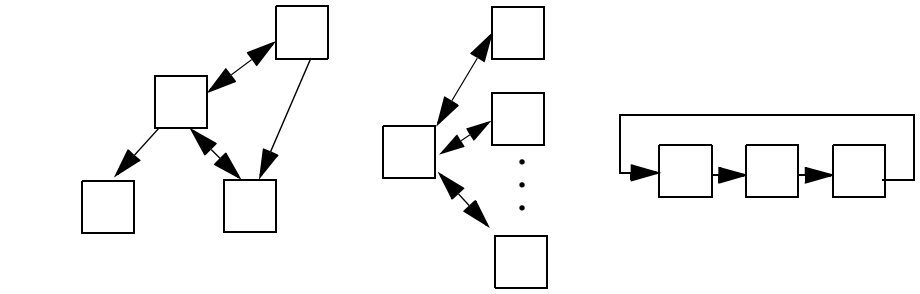


Figure 1: A meta-application consisting of five components. Three computations (A, B, C), a database server and a VR server. The presence of an arc indicates data communication.

and *pipeline* (Figure 2). Concurrent is the classic meta-application in which a set of components each running in a single site are executing concurrently and exchanging data. Examples include global climate modelling which often integrates several large-scale coupled computational models [5]. A *parallel* application is a special case of *concurrent* in which a component might benefit by distribution across multiple sites[2]. This normally applies to large-scale parallel applications in

which a task (or subcomponent) may be replicated an arbitrary number of times with minimal task interaction relative to task computation. Very-coarse grain SPMD computations or highly compute-intensive applications such as large parallel simulations, RSA factoring, are examples in this category. Finally, pipeline applications consist of a number of components connected in a chain-like fashion. Examples include certain multi-disciplinary optimization problems in which the out-



a) concurrent b) parallel c) pipeline

Figure 2: Meta-application classes.

puts of one program are often fed into another [7]. In addition, applications may be hybrids of the above classes. For example, the application may be pipelined, but one of the components may be parallel. In this work we consider concurrent and pipeline applications with components confined to a single site. Hybrid and parallel applications are the subject of future work. Scheduling meta-applications to minimize application completion time requires a cost model to evaluate the potential set of candidate schedules. These cost functions require information about the application. For each application component C_i, we assume the following information:

comp_amt [C_i] -- the amount of computation in # of instructions to be executed, C_i schedulable

comm_amt [C_i, C_j] -- the amount of communication (in # of bytes transmitted) between C_i, C_j

The following cost functions are constructed using this information and system resource information (*S* is a site, *N* is the number of components, *n* is the number of schedulable components, and *m* is the number of sites):

comp [C_i, S_k] for all ($i \le n$), ($k \le m$)

comm [C_i, C_j, S_k, S_l] for all ($i, j \le N$), ($k, l \le m$)

comm_total [C_i, S_k] for all ($i \le n$), ($k \le m$)

init_time [C_i, S_k] for all ($i \le n$), ($k \le m$)

The function *comp* gives the computation time for component C_i running in site S_k given the current state of the resources S_k is willing to provide to the application assuming no other component is scheduled there. If other components are also scheduled in S_k, then the *comp* function for these components may be degraded to reflect the sharing of S_k's resources.

The function *comm* gives the communication time spent by C_i in communication with C_j when C_i and C_j are run in S_k and S_l respectively. It has the value 0, if there is no communication. If multiple components are sharing a link, then the *comm* function for these components may be

2. This is not to be confused with a component that happens to be a parallel computation, but would not benefit by multi-site distribution.

degraded to reflect the sharing of the link. The total communication cost for a component *comm_total*, is a function of the *comm* values associated with its links. In the simplest case it would be the sum of these costs, but in other situations some link communication might be overlapped. The communication costs will vary for different component assignments due to the heterogeneity of the underlying network. For example, some sites might be vBNS-connected, but others might be limited to standard Internet connections. The function *init_time* includes any start-up overhead which could include queue time for S_k's resources, time to transmit component binaries from the initiating site if they are not already located at S_k, etc. For simplicity, we will assume it is 0 and that $n = N$ in the remainder of the paper.

Numerous research groups, including ours, are addressing the problem of producing such cost functions using application and system resource information [1][9]. Here, we focus on the problem of making scheduling decisions given these cost functions for three classes of meta-applications: pipelines in which a computation stage cannot begin until the prior stage has completed, and concurrent applications in which all component computations and inter-component communication are either overlapped or sequential. The meta-application completion time T_{CT} can be defined in terms of the component cost functions (where i, j, k, l range over the values above):

$$T_{CT}\ [\text{pipeline}] = \sum comp\ [C_i, S_k] + comm_total\ [C_i, C_j, S_k, S_l] \quad (1)$$

$$T_{CT}\ [\text{concurrent, overlapped}] = \max\ \{comp\ [C_i, S_k],\ comm_total\ [C_i, C_j, S_k, S_l]\} \quad (2)$$

$$T_{CT}\ [\text{concurrent}] = \max\ \{comp\ [C_i, S_k] + comm_total\ [C_i, C_j, S_k, S_l]\} \quad (3)$$

Other cost functions are possible depending on the application. For example, pipeline computation and communication could be overlapped, or *some* of the components within the application may have overlapped computation and inter-component communication. These cases are the subject of future work.

2.1 Scheduling Heuristic

The scheduling problem is to determine an assignment of schedulable components to site resources that minimizes T_{CT}. Unfortunately, this scheduling problem is NP-complete. An exhaustive search is not feasible since if there are n schedulable components and m sites, then there are m^n possible schedules if components can be co-located and single components cannot span multiple sites. We presume that while n may be small in practice (likely less than 10), m may grow as metacomputing environments achieve large-scale deployment. In practice, we may only want to consider a subset of sites or we may limit m to include sites that contain resources specifically requested by the application. The general steps of the scheduling process are the following:

1: determine candidate sites; for each site

2: a) collect available resources from the site

b) compute cost function estimates for each component from the site

3: run scheduling heuristic to search for best component/site assignment

Two scheduling heuristics have been developed, one suitable for compute-intensive meta-applications and the other for data-intensive meta-applications (a newly emerging class of meta-applications). In the former class, component-site scheduling is most critical, while for the latter, link scheduling (i.e. the link capacity is considered first) becomes important. Below we present the compute-intensive heuristic (the other heuristic will be presented and evaluated in the full paper). This heuristic considers components in decreasing order of computational weight:

1. order application components by decreasing *comp_amt* value
2. init PLACEMENT[C_i]= -1 for all components C_i
3. for each component C_i
   4. for each site S_k
      5. compute T_{CT} with PLACEMENT[C_i]=S_k, given PLACEMENT[C_j], j<i, unchanged
         (use Eq. 1-3 to compute T_{CT} based on application type)
      6. remember best T_{CT} and best associated site S_{best}
   7. end for
   8. PLACEMENT[C_i]=S_{best}
9. end for

This is a greedy algorithm with complexity $O(mn^2)$. There are *mn* schedules and *n* steps to evaluate T_{CT} (*n* links per component). We consider this heuristic to be scalable as *n* will be small in practice. It finds a single best candidate. The computation of T_{CT} in step 5 is based on an assignment of components up to the current component. The *comp* and *comm* terms for unassigned components are set to 0 in this calculation.

3.0 Initial Results

We have run the scheduling heuristic over a large set of simulated metacomputing environments and meta-applications, and measured the performance of the heuristic with respect to the optimal schedule. The simulated metacomputing environment consists of a number of interconnected sites that provide a particular computation and communicate rate. The communication rate between the different sites is also specified. A link variance parameter is used to vary to inter-site communication performance allowing us to simulate a truly uneven network.

Meta-applications consists of a type and a number of components. For each component we have an amount of computation and an amount of communication to all other components. These parameters are varied to allow us to simulate a wide range of application granularities. We also provide an affinity parameter which is used to bias particular components to particular sites. From this parameter, we generate an affinity value for each component site pair. This allows us to simulate environments with differing degrees of heterogeneity. This parameter is used to adjust the *comp* value. Finally, when components are collocated to particular sites, we use a parameter to degrade the sites computing power to all hosted components. This value ranges from no effect (e.g. perhaps the site has ample resources for all components) to a degradation linear in the number of hosted components. Currently, we do not degrade inter-site link performance in the event that a link is shared since we do not yet have empirical data for this parameter. For each schedule to be evaluated, the simulator can easily construct *comp*, *comm_total*, and T_{CT} from these parameter values. The function *comm_total* is defined to be the sum of the individual link communication costs for a component.

Realistic values are chosen for the parameter ranges (Table 1). For example, the intra-site link rate is based on a typical ethernet LAN, while the inter-site rate ranges from Internet WAN speeds up to reported vBNS rates. For parameters that are specified as ranges, a random value uniformly distributed over the range is generated for each simulation run.

The results that suggest that the heuristic performs very effectively over the simulated parameter ranges for the three application classes. A simulation study of over 800,000 distinct environ-

TABLE 1. Simulation Parameters

name	**value range/units**	**comments**
num sites	3 .. 10	this size covers today's testbeds
comp_rate	[1, 10000] MIPS	covers weak to powerful sites
intra_link_rate	[500, 1000] KBps -- fixed value	~ ethernet speed
inter_link_rate	[50, 100], [100, 200], [500, 10000] KBps	slow Internet up to vBNS-like
link_variance	1, 2, 5, 10	reflects ↑ network heterogeneity
affinity	1, 2, 5, 10	reflects ↑ component/site affinity
appl_type	concurrent w/wo overlap, pipeline	
num_components	3 .. 8	8 should cover most meta-apps
comp_amt	[10000, 100000], [100000, 100000], [1000000, 10000000] MInstructions	comp_ and comm_amt ranges cover medium-coarse grain apps
comm_amt	[1, 10000], [1, 100000], [1, 100000] Bytes	
col_degrade	[0, 1] -- real interval	0 = no degradation, 1 = linear

ment and application instances was performed. For each run, we execute the heuristic and search for the optimal schedule. We use several comparison metrics: how close the heuristic is to the optimal on average and the maximum variance from optimal. The first set of results are depicted in Figure 3. The heuristic performs best on the application classes in the following order: pipeline,

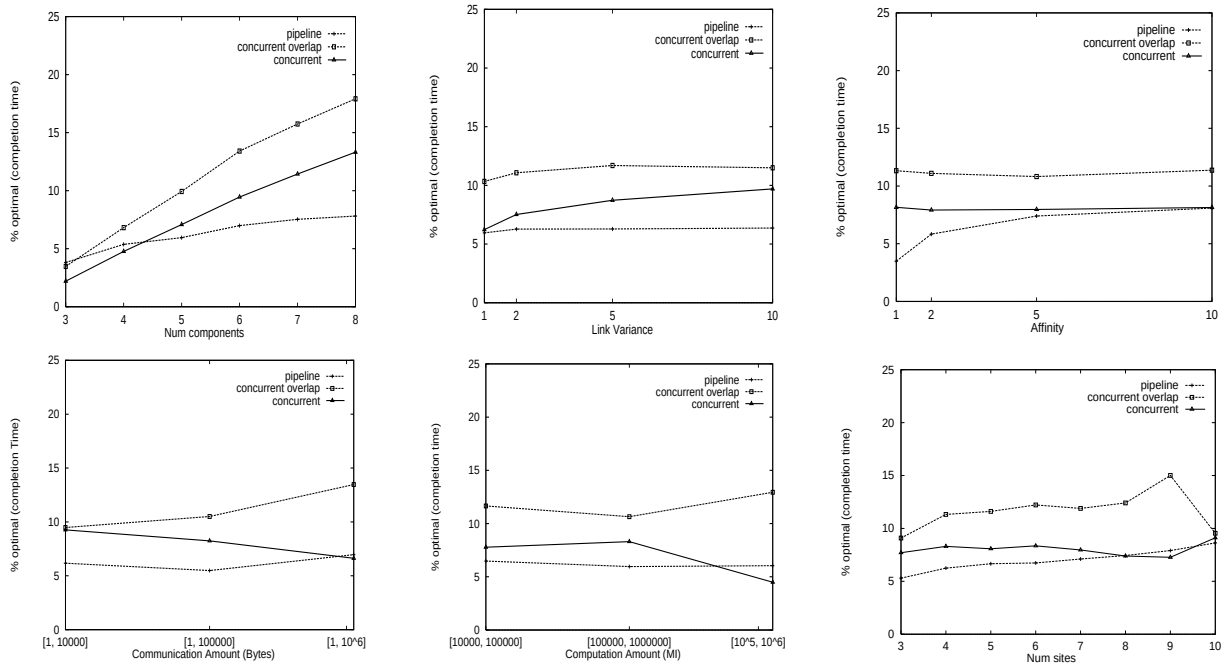


Figure 3: Performance of scheduling heuristic

concurrent, and concurrent-overlap. This order reflects increasing sensitivity of the overall completion time to the scheduling of a single component. Since the completion time of pipelines are a sum of all components, the heuristic can afford to schedule a few components sub-optimally. For the other classes, suboptimal scheduling of a single component can have a larger impact on the

completion time. However, the heuristic performs quite well for all application classes. When the number of components is ≤ 5, the heuristic is within 10% of optimal on average. In general, it is within 20% on average. The heuristic also appears to be insensitive to the heterogeneity of the network environment; performance is fairly flat for changes in link_variance and affinity parameters. The heuristic also performs well as the amount of computation and communication varies. Pipeline applications are insensitive to these parameters, while the other application classes exhibit greater sensitivity. The heuristic is also sensitive to the number of sites in the environment, but exhibits slow degradation as the number of sites increases. The second set of results indicate that the worst-case variance from optimal is within 60% for all parameter ranges, and typically within 30-40% (Table 2).

TABLE 2. Maximum variance for heuristic. Shown for each parameter value and each application class.

Parameter	**Max Variance for appl classes [% drift from optimal]: (pipeline, concurrent overlap, concurrent)**
num sites	3: (21.7, 36.5, 26.8), 4: (24.5, 41.9, 28.2), 5: (25.7, 42.7, 28.5), 6: (25.4, 45.0, 30.0), 7: (25.6, 43.4, 28.3), 8: (27.1, 47.5, 25.5), 9: (28.4, 53.5, 28.5), 10: (32.4, 42.6, 28.6)
num_components	3: (19.2, 19.4, 12.2), 4: (23.3, 32.9, 22.6), 5: (23.7, 40.4, 27.4), 6: (25.8, 49.1, 32.8), 7: (27.0, 50.7, 25.6), 8: (27.0, 57.1, 38.0)
comp_amt	r1: (25.8, 44.5, 27.9), r2: (22.8, 38.3, 28.3), r3: (27.8, 40.5, 47.1)
comm_amt	r1: (21.7, 36.5, 3.1), r2: (21.3, 35.2, 25.8), r3: (29.7, 52.6, 25.4)
link_variance	1:(25.0, 40.9, 23.4), 2: (25.5, 41.9, 27.0), 5: (23.5, 42.1, 29.9), 10: (23.1, 40.8, 32.2)
affinity	1:(14.2, 42.1, 28.2), 2: (21.0, 41.5, 27.8), 5: (29.1, 40.2, 28.0), 10: (32.7, 42.0, 28.5)

4.0 Summary

We presented a framework for scheduling meta-applications in metacomputing environments. The *metascheduling* problem is NP-complete and we developed a scalable heuristic that has performed extremely well in a simulation study of synthetic meta-applications and metacomputing environments. Completion times for three common classes of meta-applications were within 10-20% of optimal on average with a worst-case variance of 60%. The results suggest that effective scheduling of meta-applications is possible, if sufficient application and system resource cost information is provided to the system.

5.0 References

[1] F. Berman and R. Wolski, "Scheduling From the Perspective of the Application," *Fifth International Symposium on High Performance Distributed Computing*, 1996.

[2] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputing Applications,* 11(2), 1997.

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

[4] A.S. Grimshaw and W. A. Wulf, "The Legion Vision of a Worldwide Virtual Computer," *Communications of the ACM*, Vol. 40(1), 1997.

[5] C.R. Mechoso et al, "Running a Climate Model in a Heterogeneous Distributed Computer Environment," *Proceedings of the Third IEEE International Symposium on High Perform-*

ance Distributed Computing, 1994.

[6] G.J. McRae, "How Application Domains Define Requirements for the Grid," *Communications of the ACM*, Vol. 40(11), 1997.

[7] Multidisciplinary Optimization Home Page, *http://akao.larc.nasa.gov/dfc/mdo/moo.html*.

[8] J.B. Weissman, "Gallop: The Benefits of Wide-Area Computing for Parallel Processing," *Journal of Parallel and Distributed Computing*, Vol 54(2), November 1998.

[9] J.B. Weissman and A.S. Grimshaw, "A Framework for Partitioning Parallel Computations in Heterogeneous Environments," *Concurrency: Practice and Experience*, Vol. 7(5), pp. 455-478, 1995.