Modeling the Cost of Redistribution in Scheduling *

Gary Shao Rich Wolski
Fran Berman
Department of Computer Science and Engineering
University of California, San Diego

Abstract

Distributed heterogeneous resources coordinated by fast networks provide a new platform for resource-intensive parallel applications. However, careful scheduling of such applications is required to achieve their performance potential. Since resource load varies dynamically, the scheduling mechanism may consider redistribution of application tasks to improve performance. In this paper, we focus on the development of dynamically parametrizable models to determine the cost (in terms of execution delay) of redistribution. When such models are combined with predictive models for application execution, a scheduler can determine whether redistribution is profitable at a given point in time.

1 Introduction

Fast networks have enabled distributed heterogeneous computing resources to be coordinated into powerful, high-capacity parallel virtual computers. Demand for these distributed *metacomputers* is driven by resource-intensive applications that require more computational cycles, memory, and/or data transfer bandwidth than a single computer site can feasibly provide. Experience has shown that aggregation of many resources is not enough to guarantee good performance – careful *scheduling* to match application requirements with resource capabilities must be employed to achieve the best performance possible [3].

The difficulty of scheduling parallel applications is well-documented in the literature. However, scheduling metacomputer applications introduces additional complexity to the scheduling problem. In particular, a successful metacomputing strategy must account for resource *heterogeneity* as well as *contention* for shared resources. Both heterogeneity and contention cause the performance an application can obtain from system resources to vary dynamically over time. Consequently, the overall performance achievable by the application at the start of execution may also vary as the application executes. In light of this variability, there is a compelling reason to include mechanisms for modifying scheduling decisions on-the-fly in response to changes in the performance resources can deliver to the application. We denote schedulers incorporating such mechanisms as **redistributing schedulers**.

To make scheduling decisions effectively, the redistributing scheduler must be able to make an accurate estimate of the cost of the redistribution itself. If the execution delay (cost) of redistributing application tasks and data is less than the reduction in execution time (profit) that results from the new placement, the scheduler should carry out the redistribution.

*This research was supported in part by NSF Grant ASC-9308900 and ARPA order No. D570, contract #F19628-96-C-0020.

In this paper, we investigate the accuracy with which redistribution cost can be estimated for a two-dimensional data parallel Jacobi iterative solver. Note that many different redistributions may be profitable. In this paper, we assume that the scheduler has chosen a particular redistribution of the work that improves performance as measured by execution time. In Section 2, we present a formulation of a redistribution cost model that is generally applicable to periodically synchronizing data parallel algorithms. We instantiate the model using the Jacobi application in Section 3. In Section 4, we demonstrate the accuracy of the model under production computing conditions, and the potential utility of the model using synthetic workloads. In Section 5 we present related work, and we conclude in Section 6.

2 Modeling the Costs of Redistribution

The goal of a redistributing application scheduler (for a data parallel application) is to migrate data when appropriate to match new system conditions and promote improved application performance over the entire duration of a run. Redistributing application schedulers require additional capabilities not found in other forms of run-time schedulers. They must be able to evaluate the performance characteristics of an application that is already running, and they must be able to quantify the performance gains and losses produced by redistribution activities. At the point in time a redistribution cost model is evaluated by a scheduler, it must

1. estimate the performance of the application if a redistribution is not performed,
2. estimate the performance cost of a particular redistribution, and
3. estimate the performance of the application if the redistribution is carried out.

The difference between (1) and (3) represents the performance profit obtainable from a redistribution and (2) represents the performance cost.

In this paper, we focus on redistribution cost models (item (2)) for applications whose behavior can be characterized as regular repetitive cycles of computation and communication. Each cycle is associated with an iteration time; with the total execution time equal to the sum of the iteration times. We denote such applications as *iteration synchronous* applications. Although we choose execution time as the cost metric, other performance metrics can also be used within this framework. Given an application's cost metric, the goal of the scheduler is to determine an application schedule that optimizes this metric [2].

In order to have beneficial impact, a new schedule must satisfy the following relation over some range of iterations i and j:

$$T^{'}(i,j) + Redist + C_{threshold} < T(i,j) \qquad (1)$$

where

$T^{'}(i,j)$ = new execution time from iteration i to iteration j
$T(i,j)$ = execution time from iteration i to iteration j without redistribution
$Redist$ = cost of redistribution
$C_{threshold}$ = redistribution threshold

The term $C_{threshold}$ represents a threshold that the improvement must cross before redistribution is "worth it". The value of $C_{threshold}$ allows an application developer to tune the model based on the accuracy with which its parameters can be gathered. For example, if there is considerable variance in the runtime measurements that are used to parameterize the model, the $C_{threshold}$ term can be made sufficiently large so that the likelihood of performing an unprofitable redistribution is reduced to acceptable levels.

To determine *Redist* we assume that all processors begin computing simultaneously, and that overall application performance is limited by the progress of the slowest processor. Let p be a processor index. Then *Redist* is determined by four components:

$$Redist = Resched + Resync + max_p\{Remap_p\} + Restart \tag{2}$$

where

$$\begin{aligned}
Redist &= \text{time for redistribution operation} \\
Resched &= \text{time to determine a new schedule} \\
Resync &= \text{time to synchronize all processors} \\
Remap_p &= \text{time to redistribute data on processor } p \\
Restart &= \text{time to restart all computations and initialize all data}
\end{aligned}$$

Each of these components is dependent on specific characteristics of an application and the environmental conditions under which that application is run. Exact formulations for each term can only be made after the details about an application and its computing environment are clearly defined. With this in mind, we first give a general description of each term. In the next section, we will give a more detailed view of how these terms can be instantiated, using Jacobi as an example.

The *Resched* term accounts for time taken by a processor to determine alternative schedules. This activity may be distributed among processors or concentrated at a single processor. In either case, by including *Resched*, Equation 2 includes the overhead resulting from the scheduler's execution.

The *Resync* term accounts for time taken to bring all processors to a known state. For example, it may not be possible to redistribute the work until all processors have reached the same iteration number so that processors may exchange partially computed results. Some applications have processes that are tightly coupled, and therefore require no resynchronization time prior to redistribution. Other applications have processes that are more loosely coupled, allowing individual processes to proceed to later stages as long as data is available.

The $Remap_p$ term accounts for the time needed to move data to and from a processor p as a result of new data partitioning decisions. In an extreme case, all the data from one processor may have to be moved to different processor(s) before the application can begin executing again. This term is heavily dependent on the effective transfer rates of communication channels between processors, and on the size of all data blocks that must be moved. The type of transport media used also has an impact on the magnitude of the data redistribution term. In the case of shared access media like ethernet, transport operations are essentially serialized and communication times for all processors must be summed to derive total remapping time. If a switched network or other multipath transport is used, the opportunity exists for overlapping some transfers and reducing the magnitude of the redistribution term.

Restart is a term accounting for the time needed to restart an application after all the previous redistribution steps have been completed. Actions that may need to be taken include broadcast of startup commands and re-initialization of internal data structures.

3 Redistribution Cost Model for Jacobi

To illustrate how the model presented above can be applied to the development of a redistributing scheduler, we will instantiate the model components for a specific *iteration synchronous* application. We use a data parallel 2-D Jacobi solver program which operates on a square grid of double precision floating point values. In another context [3], we have developed a performance-efficient AppLeS application scheduler (which does not redistribute) for Jacobi 2D. This scheduler partitions the grid into variable-width vertical strips, which are then distributed to separate processors for calculation. We will compare a strip data partitioning of the Jacobi application with and without redistribution to determine if the cost parameterization ascribed to the redistribution model described in Section 2 is accurate.

3.1 Instantiating the Redistribution Cost Model for Jacobi 2D

Since the processors in the Jacobi application are loosely coupled, each processor can be at most one iteration ahead of any of the other processors with which it communicates. This implies that the greatest difference in iteration count between any two processors in the system is one less than the number of processors. In the worst case, for a P-processor system, the *Resync* term has a maximum magnitude of $(P-1)*E_{max}$, where E_{max} is the average iteration execution time for the processor that is farthest behind and P is the total number of processors. In the formulation of *Resync*, we estimate that the processors on average are $\frac{P-1}{2}$ iterations apart. We derive E_{max} from an application-specific model using dynamic state information [3].

In practice, resynchronization time varies as a function of how much the deviation of individual processor performance unbalances the iteration execution time. To account for this, *Resync* is modeled as proportional to the worst case time, weighted by a performance deviation variable. This variable is calculated as the ratio of the maximum positive deviation from average processor execution time to the average processor execution time. Our formulation for *Resync* for Jacobi 2D is:

$$Resync = (C_{dev})\frac{(P-1)E_{max}}{2} \tag{3}$$

where C_{dev} (the performance deviation variable) is computed as

$$C_{dev} = \min\left(\frac{max_{i=1}^{P}\{Tavg_i\} - avg_{i=1}^{P}\{Tavg_i\}}{avg_{i=1}^{P}\{Tavg_i\}}, 2\right)$$

In the definition of C_{dev}, $Tavg_i$ represents the average execution time for processor i, and is calculated by averaging the execution times from ten previous iterations. The C_{dev} deviation variable has a maximum value of 2, and reflects the influence of increasing execution time imbalance on the effective *Resync* time.

With this formulation, it can be seen that when either the iteration time or the number of processors is relatively large and processor execution time is not evenly balanced, *Resync* can be very influential in the determination of whether redistribution is advantageous or not.

The second major redistribution term of importance in Jacobi is $Remap_p$. The $Remap_p$ term must reflect the cost of transferring data between processors as well as any local management costs required by the runtime system. The formulation used in the Jacobi model for $Remap_p$ is:

$$Remap_p = Overhead_p + \sum_{i \neq j} (Xfer(i,j) * BW(i,j)) \tag{4}$$

where $Overhead_p$ is the overhead associated with any local operations associated with remapping (e.g. memory management) performed on processor p, $Xfer(i,j)$ is the amount of data being moved from processor i to processor j in bytes, and $BW(i,j)$ is the time to transfer one byte on the communication link between i and j.

This formulation was instantiated to represent the behavior of ethernet media in the network of workstations on which we ran our experiments. Redistribution in this environment is dominated by the time to transfer data from one processor to another, where each transfer can be viewed as being serialized by the shared transport mechanism. Each processor might have to wait an amount of time equal to the sum of individual transfer times contributed by other processors in the system. The additional Mem_p term is included to account for local memory management operations such as physical memory block allocation and memory-to-memory copy time.

Resched is the amount of time devoted to determining the new schedule. In the Jacobi AppLeS example, dynamic rescheduling decisions are limited to guarantee a certain minimum time between redistribution events. Since this minimum time is relatively long, application scheduler activity time can be effectively considered to be 0.

Finally, in the Jacobi application all processors are automatically resynchronized at the end of the redistribution stage. Each process in the computation runs the same application code, and there is no significant startup cost incurred after redistribution has occurred. Since there is no real contribution to execution time made by any operations of the *Restart* term, it is evaluated as 0 in this instantiation.

4 Preliminary Experimental Results for Jacobi

To help validate our formulations for redistribution costs, we incorporated the models described earlier into an AppLeS scheduling framework that was modified to include a redistribution mechanism. The goal of our experiments was to check the accuracy of the parameterization for the redistribution cost model for Jacobi 2D. To do so, we ran an AppLeS application scheduler for Jacobi 2D on eight Sun workstations that were heterogeneous in processor speeds, physical memory, network connectivity, and operating systems. All workstations were connected by ethernet, located in two separate subnets within the Computer Science and Engineering Department at UCSD. Because we were experimenting on an openly time-shared system, the experiments were exposed to contention from outside loading effects.

We conducted two experiments to measure the effectiveness of our redistribution cost model at predicting redistribution times while running in a dynamic environment. In the first experiment, we examined the accuracy of our redistribution cost model over a range of problem sizes appropriate for the platforms used in our testbed. At problem sizes less than 500x500, communication overhead costs drive the scheduler to select fewer than eight

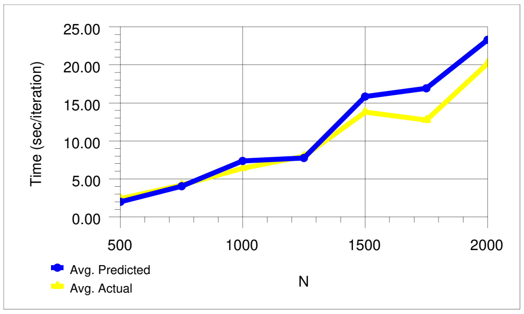

FIG. 1. *Redistribution costs with varying problem size between 500x500 and 2000x2000. The number of processors is equal to 8, the data displacement equal to 30 columns per processor, and the results were averaged from 5 trials per data point.*

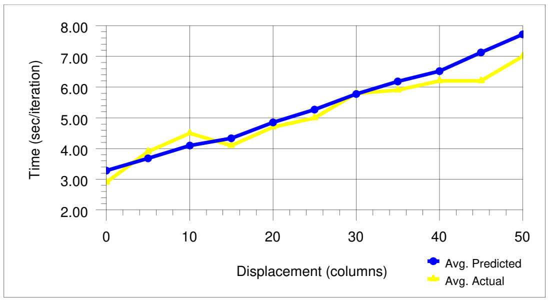

FIG. 2. *Redistribution costs with varying data displacement between 0 and 50 columns. The number of processors is equal to 8, the problem size is equal to 1000x1000, and the results were averaged from 8 trials per data point.*

processors as an optimal number to minimize execution time. At problem sizes greater than 2000x2000, limited physical memory availability in the workstations causes dramatic increases in execution time due to virtual memory paging effects. Figure 1 shows that our model was effective at predicting redistribution costs for different problem sizes while 30 columns of data were moved (displaced) during redistribution.

The second experiment examined redistribution cost prediction accuracy. We varied the amount of data displacement (in terms of the number of columns moved) as a result of redistribution while keeping the problem size fixed at 1000x1000. Figure 2 shows that our model is also effective at predicting redistribution costs over a range of data displacement sizes. Together, the two experiments provide evidence that our cost model can be effectively employed to predict redistribution costs over a range of operating conditions appropriate for the Jacobi 2D application.

In the third experiment, we examine the behavior of the cost model in the context of scheduling. We compare the performance of a redistributing Jacobi scheduler with a Jacobi scheduler which does not redistribute. The performance of the Jacobi code (estimated in Equation 1 by T and T') is predicted using the model described in [3]. This model was shown to be an accurate predictor of performance in [3]. To achieve reproducibility, we ran this experiment on a network of eight DEC Alpha workstations operating in a dedicated environment. In each trial, a "loading event" consisting of a single contending parallel program was introduced into the system while an instance of the Jacobi application was running on all eight workstations. Trials differed in the number of tasks composing each loading event. Each instance of Jacobi was allowed to run for a fixed number of iterations. In each experiment, the loading event was introduced three minutes after the Jacobi application was begun. Once started, each loading event continued until the Jacobi application had finished. Total execution times for the Jacobi application in the trials were divided by the number of iterations run to derive a measure of performance. $C_{threshold}$ was set to 15 seconds for this experiment.

In this experiment, the introduction of loading conditions was intended to simulate the effects of running in a dynamic metacomputing environment with resource contention from outside jobs. Figure 3 shows that performance for the redistributing version of Jacobi

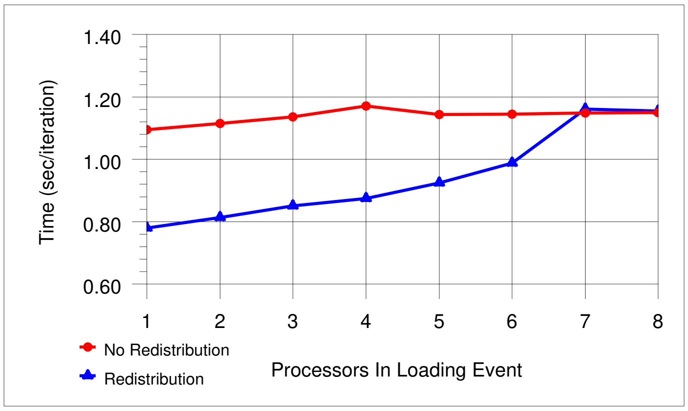

FIG. 3. *Redistributing and non-redistributing Jacobi schedulers under loading conditions simulating contention in a metacomputing environment. The number of processors is equal to 8, the problem size is equal to 2000x2000, and the experiment was run on dedicated cluster of DEC Alpha workstations.*

consistently outperformed the nonredistributing version for contending jobs executing on less than seven processors. When a contending job was introduced that required fewer than seven processors, the redistributing scheduler was able to rebalance its workload among all eight processors to maintain its performance trajectory. When the contending job executed on seven or more processors, the redistributing scheduler correctly determined that there was not sufficient performance gain attainable from reallocating the workload to justify redistribution. The redistribution cost model was successful in helping to distinguish profitable and unprofitable redistribution decisions.

5 Related Work

Scheduling parallel applications to run on heterogeneous networks of computers has attracted considerable interest in recent years. Several studies have analyzed the performance implications of resource heterogeneity and examined the potential performance gains of dynamic scheduling mechanisms [9, 12], etc. For data-parallel applications, several systems have been developed to parameterize the data decomposition of a job at run-time with dynamic application-specific and system-specific characteristics [6, 11], etc. Other systems have been developed to allow workload rebalancing to occur while an application is running [10, 4]. These systems focus primarily on redistribution to utilize idle workstation cycles, while our scheme focuses on adaptation to dynamic loads in a shared environment. SmartNet[7] accounts for system heterogeneity and resource sharing when making performance estimates and scheduling decisions, but does not support redistribution in response to changing resource loads. The PARFORM[5], Dome [1], and MARS [8] projects have investigated the use of adaptive scheduling for performance improvement in heterogeneous environments, developing systems with mechanisms for system performance monitoring and dynamic load rebalancing. However, each of these systems requires an application to either be developed within a limited programming paradigm, or to run in a special run-time environment to take advantage of rebalancing opportunities. The work we have presented in this paper is intended to reflect a more general open computing environment.

6 Conclusion

In this paper, we have defined a general redistribution model for iteration synchronous data parallel applications and instantiated its costs for Jacobi. We have provided preliminary results which indicate that the model accurately identifies the cost of redistribution for Jacobi, and that the model can be effective in an execution strategy which allows for dynamic redistribution. While it is difficult to generalize results from a single application example, the experimental results presented here show that careful accounting of redistribution costs can be used to improve application performance, and that further work to identify and quantify the costs and benefits of redistribution in a metacomputing environment is warranted.

References

[1] J. N. C. Arabe, A. Beguelin, B. Lowekamp, E. Seligman, M. Starkey and P. Stephan, *Dome: parallel programming in a heterogeneous multi-user environment*, Tech. Rep. CMU-CS-95-137, Carnegie Mellon University, (1995).

[2] F. Berman and R. Wolski, *Scheduling from the perspective of the application*, Proc. 5th IEEE Int. Symp. HPDC, (1996), pp. 100-111.

[3] F. Berman, R. Wolski, S. Figueira, J. Schopf and G. Shao, *Application-level scheduling on distributed heterogeneous networks*, Proc. Supercomputing'96, (1996), http://www.supercomp.org/sc96/proceedings/SC96PROC/BERMAN/INDEX.HTM.

[4] R. D. Blumofe and D. S. Park, *Scheduling large-scale parallel computations on networks of workstations*, Proc. 3rd IEEE Int. Symp. HPDC, (1994), pp. 96-105.

[5] C. H. Cap and V. Strumpen, *Efficient parallel computing in distributed workstation environments*, Parallel Computing, 19 (1993), pp. 1221-1234.

[6] P. E. Crandall and M. J. Quinn, *A partitioning advisory system for networked data-parallel processing*, Concurrency: Practice and Experience, 7 (1995), pp. 479-495.

[7] R. Freund, T. Kidd and L. Moore, *SmartNet: a scheduling framework for heterogeneous computing*, Proc. 2nd Int. Symp. Parallel Architectures, Algorithms, and Networks, (1996), pp. 514-521.

[8] J. Gehring and A. Reinefeld, *MARS–a framework for minimizing the job execution time in a metacomputing environment*, Future Generation Computer Systems, 12 (1996), pp. 87-99.

[9] B. Hamidzadeh, D. J. Lilja and Y. Atif, *Dynamic scheduling techniques for heterogeneous computing systems*, Concurrency: Practice and Experience, 7 (1995), pp. 633-652.

[10] J. Pruyne and M. Livny, *Parallel processing on dynamic resources with Carmi*, Proc. IPPS'95 Workshop on Job Scheduling Strategies for Parallel Processing, (1995), pp. 259-278.

[11] J. B. Weissman and A. S. Grimshaw, *Network partitioning of data parallel computations*, Proc. 3rd IEEE Int. Symp. HPDC, (1994), pp. 149-156.

[12] M. J. Zaki, W. Li and S. Parthasarathy, *Customized dynamic load balancing for a network of workstations*, Proc. 5th IEEE Int. Symp. HPDC, (1996), pp. 282-291.