Near-Optimal Adaptive Control of a Large Grid Application*

Det Buaklee   Gregory F. Tracy   Mary K. Vernon   Stephen J. Wright

Computer Science Department
University of Wisconsin – Madison

{det, gtracy, vernon, swright}@cs.wisc.edu

ABSTRACT

This paper develops a performance model that is used to control the adaptive execution the ATR code for solving large stochastic optimization problems on computational grids. A detailed analysis of the execution characteristics of ATR is used to construct the performance model that is then used to specify (a) near-optimal dynamic values of parameters that govern the distribution of work, and (b) a new task scheduling algorithm. Together, these new features minimize ATR execution time on any collection of compute nodes, including a varying collection of heterogeneous nodes. The new adaptive code runs up to eight-fold faster than the previously optimized code, and requires no input parameters from the user to guide the distribution of work. Furthermore, the modeling process led to several changes in the Condor runtime environment, including the new task scheduling algorithm, that produce significant performance improvements for master-worker computations as well as possibly other types of grid applications.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques; C.1.4[**Parallel Architectures**]: Distributed Architectures; G.1.6[**Optimization**]: Stochastic Programming.

General Terms

Measurement, Performance, Experimentation.

Keywords

Parallel algorithms, parallel application performance, stochastic optimization, adaptive computations, grid computing.

1. INTRODUCTION

This paper develops a model for near-optimal adaptive control of the state-of-the-art stochastic optimization code ATR [18] on Grid platforms such as Condor [19], Globus [10], or Legion [13], in which the number and capabilities of the distributed hosts that execute ATR varies during the course of the computation.

Stochastic optimization uses large amounts of computational resources to solve key organizational, economic, and financial planning decision problems that involve uncertain data. For instance, an approximate solution of a cargo flight scheduling problem required over 30 hours of computation on four hundred processors. To find more accurate solutions (in which a wider range of scenarios is considered), or to verify the quality of approximate solutions, may require vastly greater resources. The aim is to find the decision that optimizes the expected performance of the system across all possible scenarios for the uncertain demands. Since the number of scenarios may be very large (typically 10^4 to 10^7), evaluation of the expected performance (which requires evaluation of the performance under each scenario) can be quite expensive. ATR is also representative of a class of iterative algorithms that (1) have a basic fork-join synchronization structure, (2) require an unpredictable number of iterations to converge to a solution, and (3) can adjust the number and sizes of the tasks that are forked per iteration.

Computational grids running middleware such as Condor currently provide one of the most attractive environments for running large compute-intensive applications. These grids are inexpensive, widely accessible, and powerful. Over time, applications submitted using grid middleware are given a "fair share" of the computational resources that are not being used by higher priority computations. Applications like ATR can obtain large quantities of processing power easily and inexpensively.

To run efficiently in a grid environment, the application must be able to execute on a heterogeneous collection of hosts whose size varies unpredictably during execution. Moreover, it should be able to adapt to the changes in the size and composition of the collection of hosts, as well as to changes in the computational demands the algorithm, as it executes. It may adapt, for instance, by changing the distribution of work among the hosts.

It is unknown how to develop an adaptive version of a stochastic optimization tool such as ATR that minimizes total execution time in a grid environment. The problem is particularly complex because the parameters that govern the amount and distribution of work also affect intrinsic performance of the algorithm (such as the time to initialize each iteration and the number of iterations to reach convergence) in ways that are not easily quantified. Furthermore, the runtime environment typically includes support functions that add unpredictable delays to task execution times.

Previous work [18] in developing the ATR algorithm for Condor platforms has relied on simple task scheduling and extensive experimental measurements of total application running time as a function of (1) the average number of allocated compute nodes, and (2) the fixed (i.e., non-adaptive) values of the ATR parameters that define the number and composition of the parallel tasks in the computation. These experiments have resulted in rules of thumb for selecting the ATR parameters as a function of the number of compute nodes allocated. For example, when the

*This work was partially supported by the National Science Foundation under grants EIA-9975024 and EIA-0127857.

application runs on X ≤ 100 distributed Condor nodes, the rules provide parameters to ensure that 2X to 5X parallel tasks are available for processing at any given time, in order to keep processor efficiencies high in the presence of the high variability in the task execution times. Previous studies concerning adaptive control of distributed applications (e.g., [6][14][20]) have proposed similar kinds of experimentally determined rules of thumb. Development of adaptive codes that *minimize total execution time* for complex applications, based on more precise modeling and adaptation strategies has, to our knowledge, not previously been investigated.

This paper develops a precise and accurate performance model of ATR and then uses the model to develop an adaptive version of the code that minimizes the total execution time for a large class of planning problems, on *any* collection of compute nodes including a varying number of heterogeneous nodes. The new adaptive code does not require any user input to guide the amount and distribution of work, and implements parameter settings quite different from those obtained from the "rules of thumb" developed previously. The new adaptive code also uses (1) a higher performance task scheduling strategy, and (2) a reduced debug I/O level, and (3) a proposed simple change in the Condor runtime system support that reduces needless overhead on the master node. The latter two changes, motivated by the development of the ATR performance model, greatly improve task execution times as well as the predictability of the task execution times. All three changes can greatly improve other grid applications as well as ATR. The new adaptive code, together with the runtime system change, reduces total execution time compared to the previously optimized code by factors of six or more, depending on the planning problem and grid configuration.

The remainder of this paper is organized as follows. Section 2 provides an overview of the ATR application, the Condor runtime environment, and related work in performance of adaptive grid codes. Section 3 describes the detailed measurement-based performance analysis of ATR needed to develop the model. Section 4 describes, validates, and applies the model to select near-optimal configurations for ATR on a sizable Condor pool. Conclusions of the work are stated in Section 5.

2. BACKGROUND

Sections 2.1 and 2.2 briefly describe the ATR stochastic optimization application, the Condor system, and the MW runtime support library for master-worker computations. Section 2.3 then summarizes related work in performance modeling and development of large distributed applications.

2.1 ATR

ATR is an iterative "asynchronous trust-region" algorithm for solving the fundamental stochastic optimization problem: two-stage stochastic linear programming with recourse, over a discrete probability space. The algorithm is described in detail in [18]. Here we discuss those aspects of the algorithm that are relevant to selecting the performance-related algorithmic parameters; in particular, the parameters that control the number and composition of the parallel tasks in the execution.

The problem is as follows: Given a set of N scenarios ω_i, $i = 1, 2, \ldots, N,$ with probabilities $p_i, i = 1, 2, \ldots, N,$ solve

$$\min\ c^T x + Q(x) \quad \text{subject to } Ax = b,\ x \geq 0,$$

$$\text{where} \quad Q(x) = \sum_{i=1}^{N} p_i Q(x; \omega_i),$$

and each $Q(x; \omega_i)$ is the optimal objective value for a second-stage linear program, defined as follows:

$$Q(x; \omega_i) = \left\{ \min_{y_i}\ d_i^T y_i : W y_i = h_i - T_i x,\ y_i \geq 0 \right\}.$$

To evaluate the function $Q(x)$ thus requires the solution of N independent second-stage linear programs. When N is large, this process can be computationally expensive.

The function $Q(x)$ is a piecewise linear, convex function. The ATR algorithm builds up a lower bounding function $m(x)$ for the true objective function $c^T x + Q(x)$, using information gathered during evaluation of the second-stage linear programs.

The basic structure of the algorithm, illustrated in Figure 1, is that a master processor computes a new candidate iterate x by solving the trust-region subproblem defined below. Worker processors then evaluate the second-stage linear programs for this x and produce the information needed to refine the model function $m(x)$. The master processor updates this model function asynchronously, as information arrives from each worker processor. Outdated information may also be deleted from $m(x)$ on occasion. When sufficient new information has been received (*i.e.,* after all or most of the processors working on evaluation of x have returned their results), the master computes a new iterate x. The process then repeats.

New iterates x are generated by solving subproblems of the form:

$$\min_x m(x) \quad \text{subject to} \quad Ax = b, x \geq 0, \left\| x - x^I \right\|_\infty \leq \Delta$$

where x^I is the "incumbent" (the best iterate identified by the algorithm to date), while Δ is the "trust-region radius," which defines the maximum distance we can move away from the incumbent on the current step.

In general, the more scenarios N that can be included in the formulation, the more realistic is the model. Although the problem becomes larger and harder to solve as N increases, this effect is less marked if we have a good initial guess of the solution. A good strategy is therefore to start by solving an approximate problem with a modest value of N (1000, say), and use the resulting approximate solution as the starting point for another approximate problem with a larger number of scenarios (5000, say). This procedure can be repeated with progressively larger N.

To reduce the time to compute each new iterate x, the N scenarios are partitioned into a fixed set of T tasks denoted by $N_1, N_2, \ldots N_T$ where each N_j denotes a set of scenario indices. For a given task j a worker computes the following partial sum of $Q(x)$:

$$Q_{[j]}(x) = \sum_{i \in N_j} p_i Q(x; \omega_i).$$

The worker also computes one partial subgradient (also known as a "cut") for the task. This information is used by the master to update the partial model function $m_{[j]}(x)$ corresponding to task j.

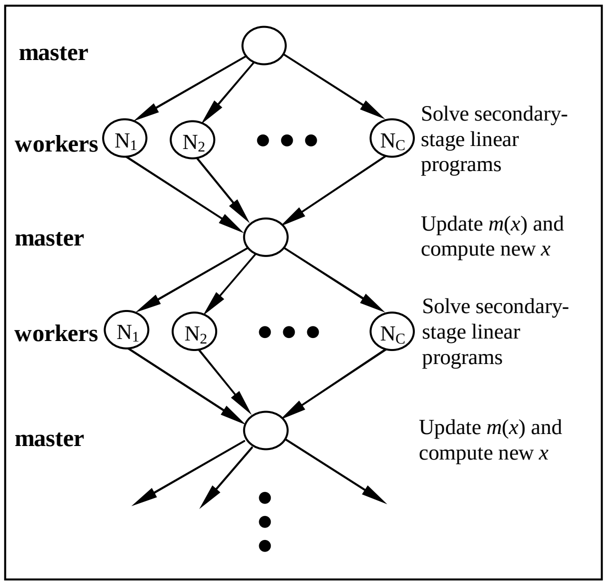

master
workers
N_1
N_2
N_C
Solve secondary-stage linear programs
master
Update $m(x)$ and compute new x
workers
N_1
N_2
N_C
Solve secondary-stage linear programs
master
Update $m(x)$ and compute new x

Figure 1: Basic ATR Structure (B=1)

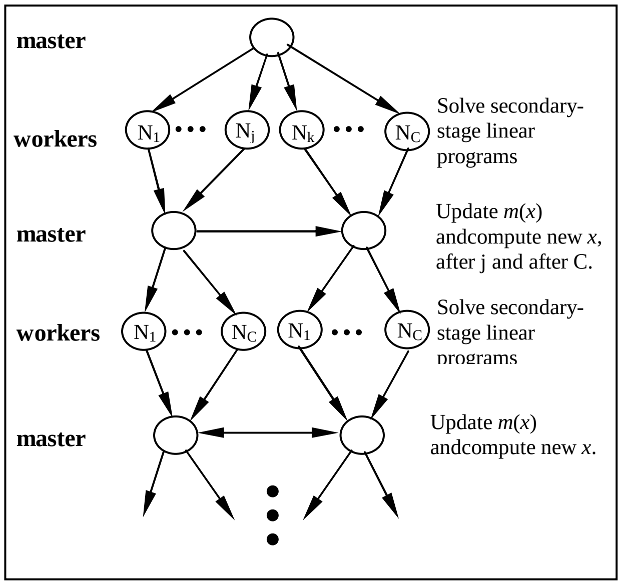

master
workers
N_1
N_j
N_k
N_C
Solve secondary-stage linear programs
master
Update $m(x)$ andcompute new x, after j and after C.
workers
N_1
N_C
N_1
N_C
Solve secondary-stage linear programs
master
Update $m(x)$ andcompute new x.

Figure 2: Asynchronous ATR Structure (B=2)

The complete model function $m(x)$ is then $c^T x$ plus the sum of the $m_{[j]}(x)$'s over all the tasks $j = 1,2,...T$.

ATR enables additional parallelism by allowing more than one candidate iterate *x* to be evaluated at the same time. In order to generate an additional candidate iterate, the master processor computes the new iterate before all of the scenarios for the current candidate have been evaluated, as illustrated in Figure 2. ATR thus creates and maintains a *basket* of *B* candidates (with *B* between 1 and 15).

In the previous non-adaptive version of ATR, tasks are grouped into *G* equal-size task groups, each containing *T*/*G* tasks. Each task group is a unit of work that is sent to a worker. For example, a possible configuration for *N* = 10,000 is *T* = 100 and *G* = 50, so that each task group contains two tasks, each with 100 scenarios.

We assume *N* to be fixed in our performance analysis. We can affect the amount and distribution of work, by varying the parameters *B*, *T*, and *G*. By increasing *B*, new iterates can be solved on the workers while the master is processing the results from other iterates, and a slow worker will only slow down the evaluation of one of the parallel iterates. By increasing *T*, more cuts are computed per value of *x*, making the model function $m(x)$ more expensive to compute (by the master) but also a better approximation to the true objective $Q(x)$, which generally reduces the number of iterations needed to solve the problem. By increasing *G*, we obtain more task groups, with fewer tasks (and therefore smaller execution times) for each group.

Previous evaluations of ATR on locally and widely distributed Condor-MW grids have shown that for 50-100 workers, the best execution times were obtained with three to six concurrent candidates, 100 tasks, and 25 to 50 task groups.

The goal in this research is to create a performance model to select the basket size *B*, the number of tasks, *T*, and the number of task groups *G*, statically at program initiation time or dynamically during the execution of the job, so as to *minimize* total ATR execution time for any given planning problem of interest.

2.2 Condor/PVM and MW

The Condor system [19] manages heterogeneous collections of computers, including workstations, PC clusters, and multiprocessor systems. Mechanisms such as glide-in or *flocking* [9] are used to include processors from separate sites in the resource pool. When a user submits a job to the system, Condor identifies suitable processors in the pool and assigns the processors to the job. If a processor executing the job becomes unavailable (for example, because it is reclaimed by its owner), Condor migrates the job to another node in its pool, possibly restarting from a checkpoint that it saved at an earlier time. The size of the pool available to a particular user can change unpredictably during a computation, although users can exercise some control over the resources devoted to their application by specifying the number, speed, and/or type of workers.

MW [18] is a framework that facilitates implementation of master-worker applications on a variety of computational grid platforms. In this study, we use the version of MW that is implemented for Condor-PVM [32], in which the master runs on the submitting host, and the worker tasks on other processors drawn from the Condor pool. Condor-PVM primitives are used to buffer and pass messages between master and workers.

2.3 Related Work

Parallel stochastic optimization algorithms have been investigated during the past 15 years. In [4], an algorithm related to ATR (but without the trust region and asynchrony features) is applied to multistage problems, and implemented on a cluster of modest size. An earlier paper [5] describes an interior-point approach in which the linear algebra computations are implemented in parallel, but this approach does not scale well to a large number of processors. A small PC cluster is used in [11], where the approach is to use interior-point methods for the second-stage problems inexactly and an analytic center method for the master problem.

Three approaches to adaptive software control have been explored in previous work, namely (1) provide an interface for the user to control application steering parameters (e.g., [2]), (2) use measured or user-provided estimates of application execution time as a function of system configuration, to heuristically adapt system resource management policies or application configuration parameters, automatically at runtime [17][6][14][16][20][27], or (3) use very simple models that compute execution time for alternative configurations [3][8][21][23][24][25] from user-provided or measured deterministic processing and communication requirements per task, possibly adjusted for runtime-measured node processing capacities and available network bandwidth.

Regarding history-based heuristic adaptive control algorithms, Ribler et al. [20] describe the Autopilot system that filters data from instrumented client tasks (e.g., to characterize the dominant file access pattern) and adapts the system resource management policies (e.g., file prefetch policy). The related GrADS Project work [27] measures the "application signature" (i.e., processing, I/O, and communication cycles as a function of time), and uses fuzzy logic to determine whether the rates for each task are within ranges defined by user-specified "performance contracts". Algorithms for changing the configuration when the contract is violated are not addressed in that work. Heymann et al. [14] measure Condor-MW task execution times, and worker node efficiencies, in each iteration. Results from synthetic MW applications are used to estimate the number of worker nodes to allocate to achieve 80% efficiency with no more than a 10% increase in iteration execution time, based on the relative processing times of the largest 20% of the tasks. This algorithm assumes each task performs (approximately) the same work in successive iterations, homogeneous processing nodes, and that the application completes eachiteration before starting the next iteration. Lan et al. use computation times measured in previous iterations to decide whether to redistribute work in the next iteration of an astrophysics code [17]. Chang and Karamcheti [6] propose an application structure containing a "tunability interface" and an expression of user preferences. Previous application execution measurements, runtime resource and application monitoring, and user preferences are used to automatically select certain parameters at runtime, for example to select the image resolution for the available processing capacity and a user-specified image transmission time. In the Harmony system, Keleher et al. [16] propose an approach which the user specifies the processing and communication time, or provides a model to predict these values at runtime, for each possible multidimensional configuration of the application. The system then dynamically allocates resources to achieve a particular system objective, such as maximizing throughput.

Table 1: ATR Configuration Parameters

Symbol	Definition
N	Number of scenarios evaluated per iterate
T	Number of tasks per iteration
G	Number of task groups into which scenarios are partitioned
x	Vector of candidate planning decisions; "iterate"
B	Number of iterates that are evaluated in parallel

Previous models for adaptive runtime control compute node processing time *and* communication time per iteration, using known and/or measured quantities per data point, or image pixel, times the number of data points or pixels assigned to the node. For example, Ripeanu et al. [21] compute the execution time for a load-balanced finite difference application as a function of (a) the data assigned to each node, (b) redundant work computed by other nodes to reduce communication between nodes, and (c) the communication time. These calculations are used at runtime to select the amount of redundant work per node for the measured Grid communication costs. The AppLeS project has used similar calculations to determine how many and which available compute or data server nodes should be assigned to a simple adaptive iterative Jacobi application [3], a gene sequence comparison code [24], a magnetohydrodynamics application [22], an adaptive parallel tomography image reconstruction application [23], and adaptive data server selection in the SARA application [25]. For each of these applications, a linear optimization model is formulated to compute the work assigned to each node per candidate system configuration and measured node processing and communication capacities, to achieve an objective such as minimizing total execution time or maximizing image quality for a user-specified target refresh frequency.

The approach in this paper is most similar to the model-based adaptive runtime control in [23]. However, we are targeting the much more complex ATR application that does not have a known model for estimating execution time as a function of system configuration (*i.e.*, parameters B, T, G, and set of workers). Furthermore, we develop a model that more efficiently determines how to allocate work to available compute nodes without solving an optimization problem over all possible system configurations.

3. ATR EXECUTION TIME ANALYSIS

In this section we analyze ATR task execution times and communication latencies to determine which aspects of the computation and communication need to be modeled, and how to model the total ATR execution time. The analysis is guided by the task graphs in Figures 1 and 2, which illustrate the overall structure of a parallel ATR execution, for basket size B equal to one and two, respectively. (Table 1 summarizes the notation.) The basic functions of the master process are to (a) update the function $m(x)$ each time a worker processor returns the results of executing a group of tasks, and (b) compute a new iterate x when all T tasks associated with a particular iterate x have been completed.

In the existing *non-adaptive* version of ATR, the parameters B, G and T are specified as inputs to the application and are fixed throughout the run. G specifies the number of task groups, which can be evaluated in parallel (by the workers). Each task group contains T/G tasks each containing N/T scenarios. Since each individual task generates a subgradient (or cut), each task group returns T/G subgradients.

The goals of this work are to determine:

- whether the values of B, G, T, and the number of tasks per group can be determined so as to minimize total ATR execution time on a collection of workers,
- whether near-optimal *adaptive* values of the parameters can be computed at runtime as the collection of workers changes, and

Table 2: Example Measured Execution Times
(SSN, N=10,000)

G	T/G	T	Master Time to Update Model Function *m*(*x*) (msec)				Master Time to Compute a New Iterate, *x* (sec)					Worker Execution Time (sec)	
			avg	min	max	CV	num it.	avg	min	max	CV	avg	CV
25	1	25	6.51	3.36	915.00	4.81	82	0.38	0.01	2.06	1.00	20.54	0.08
50	1	50	6.04	3.56	1405.46	5.71	47	1.32	0.01	3.05	0.68	10.56	0.02
50	2	100	6.83	3.64	1936.09	0.79	32	2.42	0.05	7.60	0.79	10.36	0.06
100	1	100	5.94	3.40	1162.59	40.20	31	1.57	0.05	3.41	0.83	5.19	0.05
200	1	200	6.12	3.84	2092.39	4.89	25	2.25	0.03	6.25	0.71	2.69	0.06
400	1	400	6.74	3.30	2411.61	3.92	21	3.33	0.05	13.27	0.75	1.35	0.08

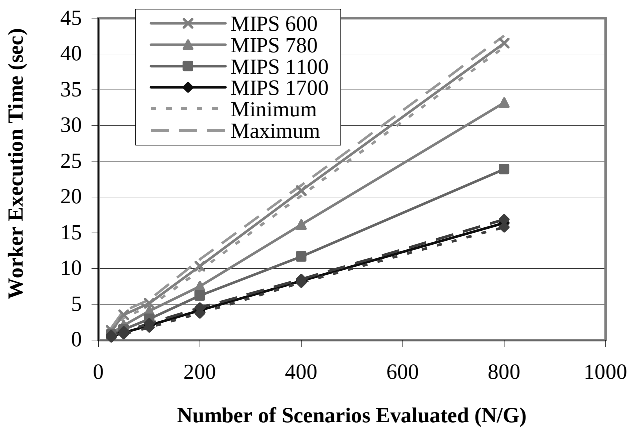


Figure 3: Worker Execution Time
(SSN Planning Problem, *N*=10,000, *B*=1, *T/*G=1)

- how much performance improvement can be gained from the adaptive version of the code.

The challenge for modeling and minimizing the overall ATR execution time is that previous measurements of ATR executions [18] have revealed that it is difficult to quantify (a) how master execution times increase as *T* increases, (b) how the number of iterates that need to be evaluated increases as B increases or T decreases, and (c) the high degree of *unpredictability* in the execution times of the master and workers, which is possibly due to the runtime environment. These issues are addressed below; Section 3.1 analyzes worker execution times, while Section 3.2 analyzes master execution times. Section 3.3 analyzes Condor-PVM communication costs between a pair of local hosts, as well as for a pair of widely separated hosts, for message sizes that are transmitted between the master and a worker in the ATR application. These measured computation and communication times are used in Section 4 to develop a performance model and optimized adaptive parameter values for the ATR code.

A standard approach in our local Condor pool is to submit a Condor job from a shared host, which becomes the master processor. Because the shared host typically has a relatively high CPU load, master processing times might be highly variable and/or large. To avoid this problem, unless otherwise noted, the ATR measurements reported in this section are submitted from a single-user workstation that is free of other user processes during the run. We refer to this setup as a "light load" master processor.

The initial analysis of execution times is based on several planning problems, several values of *N* (numbers of scenarios), a basket size *B* = 1, a wide range of values of *T* and *G*, and a *single* worker node. After analyzing the impact of *T* and *G* with *B* = 1, we investigates larger values of *B*. The use of a single worker ensures that the measured master task execution times are not inflated by unpredictable interrupts from workers that have finished evaluating other tasks. Once the basic task execution times are understood, interrupts can be modeled as needed.

3.1 Worker Execution Times

Table 2 summarizes the average, minimum, maximum, and coefficient of variation (*CV*) in the execution time for the two principal tasks carried out by the master, namely, updating *m*(*x*) and computing a new iterate. The table also shows the average and CV of the time for a worker to evaluate a task group, over all of the iterations, for several different values of *G* and *T*. Similar results were also obtained for other planning problems, other values of the number of scenarios *N*, runs at different times of the day, and for many different worker processors.

The measurements show that worker execution times in each experiment have low variability (i.e., are highly predictable). Furthermore, results in Figure 3 show that, in the common case that *N*/*G* ≥ 25, the average worker execution time is approximately linear in the number of scenarios evaluated, *N/G*, as well as in the peak speed of the processor. The second and third rows of Table 2 (as well as other results omitted to conserve space) demonstrate that worker execution time is independent of *T*.

The above results indicate that the execution time for a given ATR task on a given worker can be predicted from a benchmark that contains at least 25 scenarios, which can be run on the worker when it is first assigned to the ATR application. We note that the deterministic worker execution times are due to the fact that the Condor job scheduler uses space-sharing, rather than time-sharing, for grid nodes that are reasonably well utilized. As a consequence, interference from other jobs need not be modeled in predicting ATR execution times on Condor.

3.2 Master Execution Times

Table 2 shows that the master processing times, both for updating *m*(*x*) and for computing a new iterate, are *highly variable*. It might

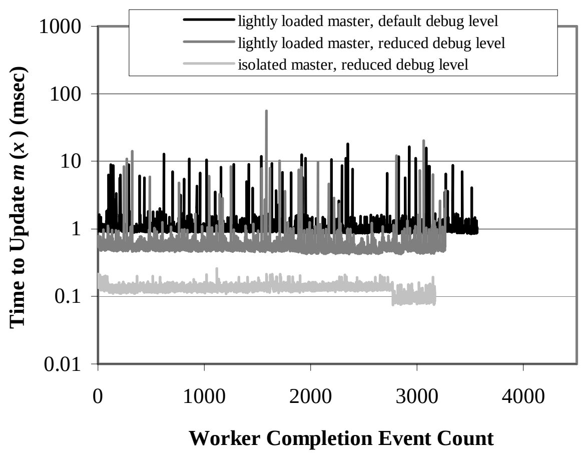


Figure 4: Example Histogram of Times to Update m(*x*)
(SSN Planning Problem, *N*=10,000, *B*=1, *T*=50, *G*=50)

appear that the variability in the time to update *m*(*x*) is not important, since the average time required for each update is only a few milliseconds. However, since the master needs to perform this update *G* times per iteration (each time a worker returns the results from evaluating a task group). Since *G* may be 100 or more, and since the largest update times are on the order of 1-2 seconds (as shown in Table 2), the cumulative update time can have a significant impact on total ATR execution time. In Section 3.2.1 we (a) analyze the causes of the execution time variability; (b) propose two changes in the runtime system to reduce the variability; and (c) characterize this task execution time more precisely. We note that experimentally observed variability in worker execution times in previous work may actually have been due to variability in the master processing time for updating *m*(*x*).

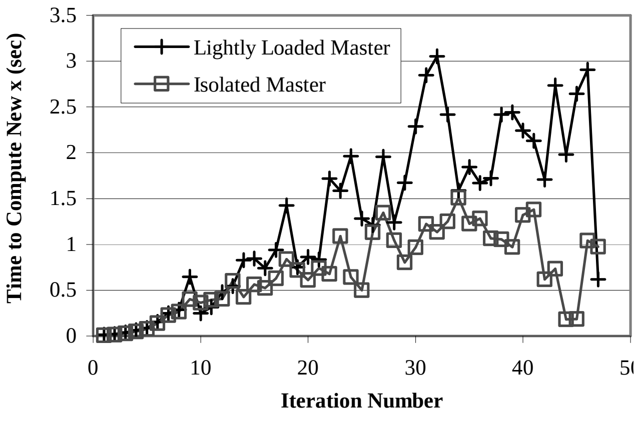
As noted in Section 2.1, as *T* increases, the average time to compute a new iterate increases, while the number of iterations decreases. This is because *T* cuts are added to the model function *m(x)* at each iteration, so a larger *T* causes the model function to become a closer approximation to the true objective function after fewer iterations, while making each trust-region subproblem harder to solve. The time to compute the new iterate varies between under one second and three to four times the average value. We investigate these variations in more detail in Section 3.2.2 with the goal of understanding how to model these variations as a function of *G* and *T*.

3.2.1 Time to Update m(x)

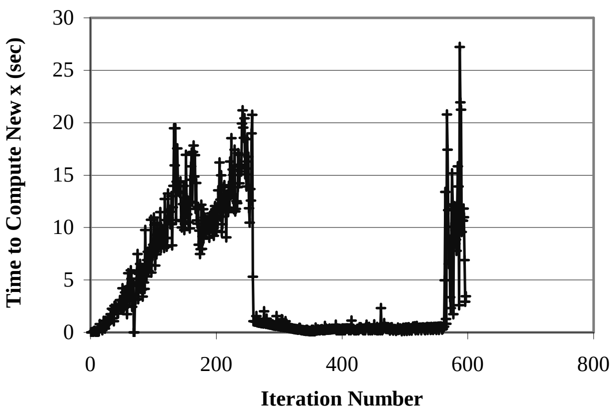
Figure 4 provides example histograms of the master execution times for updating the model function *m*(*x*) during three different measurement runs. The histogram for the lightly loaded (700 MIPS) master with default debug I/O corresponds to the data in Table 2 for *G* = 50 and *T/G* = 1. Even greater variability than shown in this histogram is observed if the master is one of the shared Condor hosts commonly used to submit Condor jobs.

Further investigation revealed that the value of the Condor "debug I/O flag" commonly used for MW applications produces vast quantities of debug output. Figure 4 provides a second histogram ("lightly loaded master with reduced debug I/O") for a run with a *reduced level of debug I/O* (level 5) from Condor/MW, which still produces a significant log of the MW execution events. The reduced debug I/O improves the average time to update *m*(*x*), but not the variability in the execution time.

To understand the high variability in the observed execution times, we note that due to the worker execution times (see Table 2), there can be significant periods of time (on the order of seconds) when the ATR master is waiting for results from the workers. Since the master processor is part of the Condor pool, we surmised that the high variability in the time to update the model function shown in Figure 4 is at least partly due to Condor administrative functions or to other Condor jobs that may run on the master processor during these periods. The third histogram in Figure 4 shows that using an *isolated* master node, which is not available for running other Condor jobs or administrative functions during the ATR run, greatly reduces the variability as well as the average execution time to update *m*(*x*). Thus, we proposed a new feature for Condor that allows a user-owned host serving as the (lightly loaded) MW master to be "temporarily unavailable" for running other Condor jobs.



(a): Impact of Isolated Master
(SSN Planning Problem, *N*=10,000, *B*=1, *T*=50, *G*=50)



(b): Typical Profile (Isolated Master)
(20-term Planning Problem, *N*=5,000, *B*=1, *T*=200, *G*=50)

Figure 5: Example Histograms of Times to Compute a New Iterate, *x*

For the isolated master configurations, the average time to update *m*(*x*) is 150-500 *microseconds*, depending on the value of *T*/*G*. In this case, the overall ATR execution time is dominated by the worker execution times and the time for the master to compute new iterates (see Table 2).

3.2.2 Time to Compute a New x

Figure 5 shows a histogram of the time to compute the next iterate, for a given planning problem and a given set of input parameters. Figure 5(a) shows that an isolated master exhibits lower mean and variability in execution time for computing the new iterates than the same master in a lightly loaded (non-isolated) configuration. The lower variability makes it easier to optimize the application parameters that control parallelism. Except as noted, all measurements of ATR below are obtained with the isolated master and the reduced (but still substantial) debug I/O level.

Figures 5(b) and 6(a) show histograms of the execution times for computing new iterates *x*, for the two very different planning problems 20term and SSN. In Figure 5(b), the starting point was "blind" (that is, far from the solution), while in 6(a) it was chosen as the solution of the corresponding planning problem for a smaller number of scenarios *N*. In both figures, we see a wide variability in the time required to compute the next iterate from one iteration to the next, but the trend is similar. Specifically, the time required tends to increase steadily, then drops off sharply and becomes minimal for a (possibly long) sequence of iterates. The times increase again for the last few iterations. Similar trends were also observed in other planning problems, and many other starting points and parameter settings.

An ad hoc adaptive algorithm might estimate the average time to compute the next few iterates from the time to compute the current iterate (with fairly high accuracy in most iterations). It could use this estimate to adjust the parameters governing parallelism in evaluation of the current iterate(s) (namely, *B* and *G*) so as to achieve a desired trade-off between processor efficiency and expected total time during the next iteration.

We develop an alternative approach in Section 4 that is based on the following observations. Figures 6(a)-(c) illustrate the dependence between the parameter *T* and the number of iterations and time required to compute each new iterate, for the SSN planning problem and a fixed starting point. Note that increasing *T* causes (a) a diminishing decrease in the number of iterations, and (b) a diminishing increase in the time to compute each new iterate. The overall effect, illustrated in Figure 6(d), is that the total master processing time for computing new iterates grows

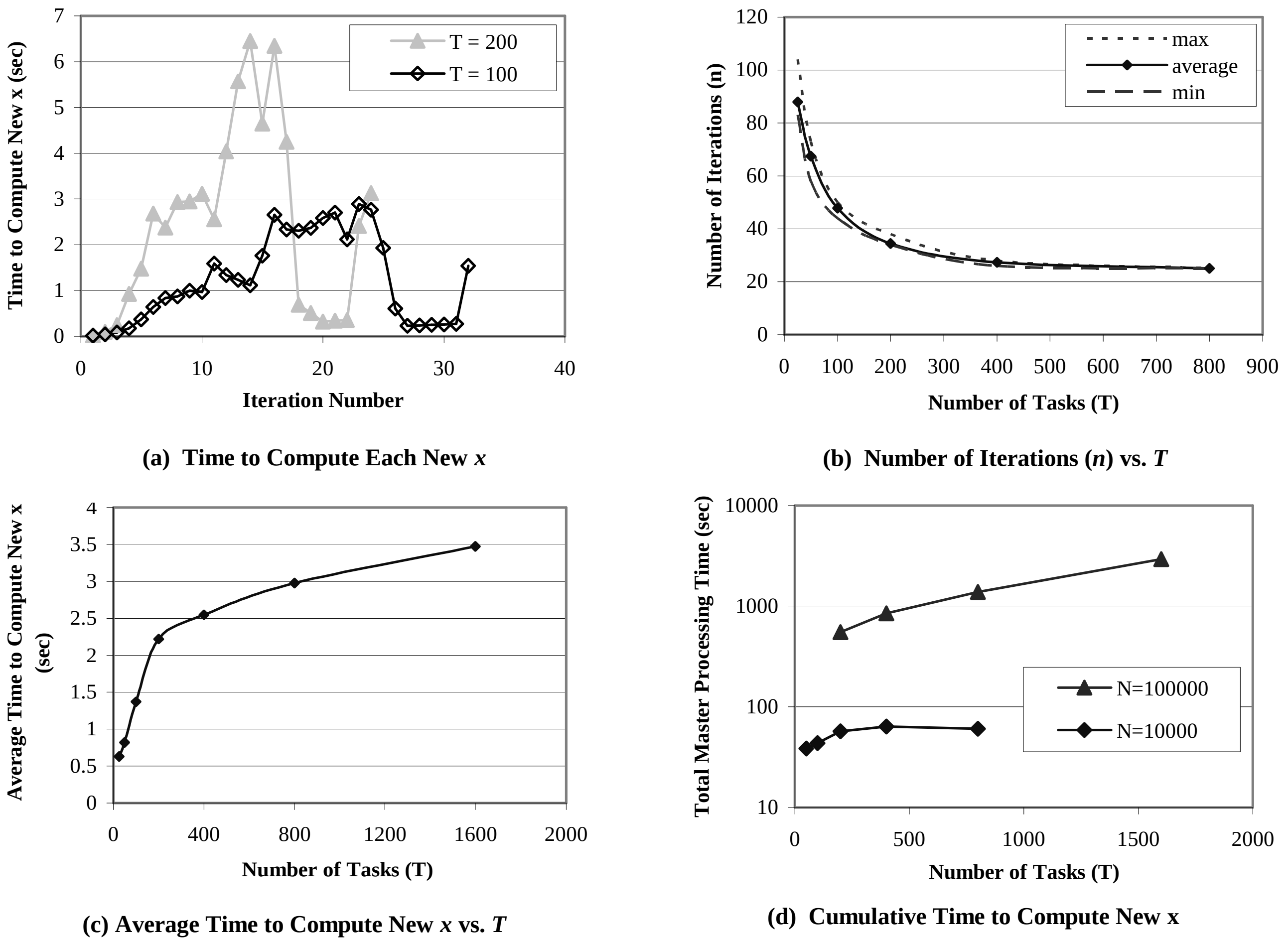


Figure 6: Impact of *T* on Time to Compute New *x* (SSN Planning Problem, *N*=10,000)

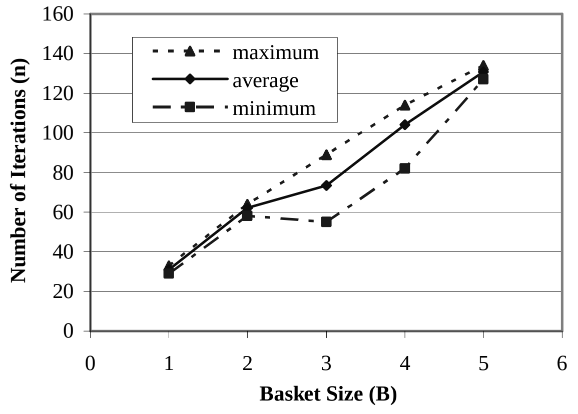


Figure 7: Impact of *B* on Total Number of Iterations (*n*)
(SSN Planning Problem, *N*=10,000, *T*=200, *G*=50)

slowly with *T*. Other curves omitted for clarity show that the time to compute the next iterate is largely independent of *G*. These observations, which hold for the three planning problems and all ATR parameter values that we analyzed, are the key insights needed to optimize *B*, *G*, and *T* in Section 4.

3.2.3 Impact of Basket Size (*B*)

By setting $B = 2$, the workers can evaluate the scenarios corresponding to one iterate *x*, while the master computes a new value of *x* from the latest subgradient information. Since, for practical planning problems, the master execution times tend to be either under one second or at least as large as the worker execution times, *B* should be set to at most two in order to improve overall execution time. However, Figure 7 shows that the total number of iterates *n* required for convergence of ATR nearly doubles as *B* doubles. (Similar behavior was observed for other planning problems and many values of *N*.) Thus, the total execution time will not improve for *B* greater than one, a fact that we have verified experimentally.

3.3 ATR Communication Times

Figures 8(a) and (b) provide the measured CondorPVM round trip time for sending a message of a given size to another host and receiving back a small message, a process that mimics the round-trip communication between the master and a worker in ATR. Figure 8(a) shows the results for hosts interconnected by a high speed local network; Figure 8(b) provides the results for a host at the University of Wisconsin sending to a host in Bologna, Italy.

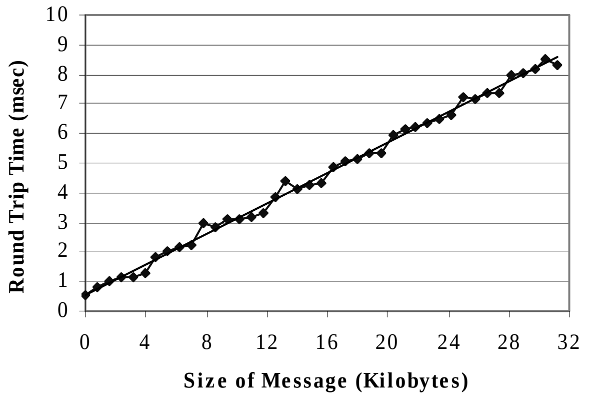
For ATR planning problems, the size of the message from the master to the worker typically ranges from 250 bytes to 12 KB. Furthermore, sending a message to a given worker is overlapped with the execution time of the worker that received the previous message. Thus, there is approximately one round-trip time per iteration in the critical path of an ATR computation. A comparison of the round trip time, worker execution times, and master execution times suggests that communication costs are negligible for practical values of *N*, *G*, and *T*, even over wide area networks.

4. NEAR-OPTIMAL ADAPTIVE ATR

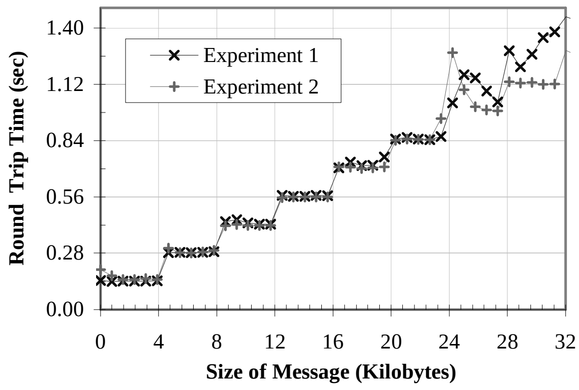
The measurements reported in Section 3 have provided the data needed to create a model of total ATR runtime. To summarize:

- Worker execution times are deterministic, approximately linear in the number of scenarios per task group (*N*/*G*), and independent of the number of tasks in the group (*T*/*G*).
- Communication costs between the master and workers are negligible (over a high speed network), even when the master and workers are widely distributed. We validate this again below for an ATR run on a widely distributed Condor flock.
- The master processing time for updating the model function, *m*(*x*), each time a worker returns its results, is 100-500 msec and can be omitted in a (first-cut) model of total ATR run time.
- The time required for the master to solve the trust-region subproblem to compute each new iterate, *x*, is a significant component of total ATR execution time. Moreover, the total master execution time to compute all iterates increases slowly with *T*, while there is a significant decrease in the number of iterates (n) that need to be considered as T increases, up to about *T*=400 (or until *N*/*T* decreases to a few tens).

These observations motivate a surprisingly simple first-cut model of the ATR execution time that can be used to optimize the execution of ATR on various grid configurations. Because



(a) Between Local Nodes



(b) Between Wisconsin and Bologna, Italy

Figure 8: Roundtrip Message Time

Table 3: Simple Model Estimates of Total ATR Execution Time for Homogeneous Workers

Planning Problem	N	T	G	Number of Workers	Compute New x (sec)		Benchmark Avgerage (t_W) (sec)	Total Execution Time (min)		Note
					num it. (n)	Total (t_M)		Model	Measured	
20-terms	5,000	200	50	G	597	2762.94	2.35	69.47	70.54	WI pool
ssn	40,000	100	50	G	84	297.36	30.97	48.83	52.21	WI-NM Flock
ssn	20,000	50	50	G	108	180.90	20.91	40.84	44.70	WI-Argonne Flock
ssn	20,000	100	50	G	84	244.00	20.89	33.51	36.38	WI-Argonne Flock
ssn	20,000	200	50	G	61	295.30	20.88	26.40	29.32	WI-Argonne Flock
ssn	5,000	200	50	G	131	1076.82	3.23	25.05	26.80	WI pool
ssn	10,000	25	25	G	104	65.43	12.95	23.75	26.77	WI pool
ssn	20,000	400	50	G	44	441.80	20.96	22.98	24.98	WI-Argonne Flock
ssn	10,000	50	25	G	73	59.74	12.85	16.84	18.60	WI pool
storm	10,000	200	50	G	11	2.53	82.44	16.53	18.48	WI pool
ssn	10,000	100	25	G	49	65.51	12.92	11.86	13.27	WI pool
ssn	10,000	50	50	G	66	56.70	6.44	8.14	9.23	WI pool
ssn	10,000	200	25	G	30	66.65	12.82	7.73	8.80	WI pool
ssn	10,000	100	50	G	50	71.79	6.48	6.70	8.23	WI pool
ssn	10,000	200	50	G	38	79.46	6.44	5.51	6.62	WI pool
ssn	10,000	400	50	G	26	70.45	6.43	4.07	4.92	WI pool
ssn	10,000	100	100	G	44	70.32	3.31	3.65	4.71	WI pool
ssn	10,000	100	500	2G/3	44	64.81	6.32	10.34	12.10	WI pool

interrupts, communication costs, and variability in task execution times are not modeled, the model is even simpler than the LogGP type of model than has been used for other large complex applications [1, 26]. Section 4.1 presents the model and validates that it is sufficiently detailed to estimate overall ATR execution time, on widely distributed Condor flocks as well as on a local Condor pool. Sections 4.2 and 4.3 demonstrate how the model, together with improved task scheduling, can be used to minimize ATR execution time on a varying set of homogeneous grid nodes and a varying set of heterogeneous grid nodes, respectively.

4.1 Model Validation

For fixed values of *N*, *G*, and *T*, and a *homogeneous* set of *G* worker nodes, a first-cut model of total ATR running time is simply $t_M + nt_W$, where t_M is the total master execution time for computing new iterates, *n* is the number of iterations, and t_W is the time needed by a worker to evaluate a group of tasks containing *N*/*G* scenarios. Table 3 evaluates the accuracy of this model, which ignores interrupts, communication overhead and small master execution times to update m(x). The table compares measured ATR execution times against execution time computed from the model using the measured components (t_M and t_W), for several different configurations of the planning problem SSN, as well as two representative versions of the problems Storm and 20-term. The experiments in the table were run on a local Condor pool or one of two Condor "flocks" in which the master processor is a local (isolated) master while the homogeneous workers are compute nodes at the Albuquerque High Performance Computing Center or at Argonne National Laboratory. In the flock experiments, communication between the master and the workers is via the Internet, so communication costs are more similar to those graphed in Figure 9(b) rather than those in Figure 9(a).

Table 3 shows that over a wide range of total application execution times, from just a few minutes to over an hour, the runtime estimates obtained from the simple model are within about 10% of the measured execution times, even when the workers are geographically distant from the master.

If we employ fewer workers *K* than the number of task groups *G*, then the model of the non-adaptive ATR running time is modified as follows: $t_M + nt_W \lceil G/K \rceil$. The last row of Table 3 (and other similar experiments omitted to conserve space) validates this simple extended model, showing that it captures the principal components of total run time.

For fixed *N*, *G*, and *T*, and a set of *heterogeneous* workers, the total execution time is estimated by $t_M + nt_W^{\max}$. Table 4 validates this model for collections of heterogeneous nodes from our local Condor pool. We obtained these results by requesting *G* workers, without restricting the type of worker nodes assigned. In this case, Condor allocated a wide variety of processors, ranging in speed from 186 MHz to 1.7 GHz. For the non-adaptive version of ATR, the total execution times estimated are generally as accurate as for the homogeneous workers, unless the problem size is fairly small (i.e., execution time is less than 10 minutes) and the number of workers *G* is large. In these cases, the worker tasks are

Table 4: Predicted and Measured Total ATR Execution Time on Heterogenous Workers

Worker Time (t_W) (sec)			Non Adaptive Execution Time (min)			Adaptive Execution Time (min)			Estimated Speedup (%)	Problem Size (N)
avg	min	max	Measured	Model	Number of Workers	Measured	Model	Number of Workers Used		
11.52	4.22	42.39	48.75	47.35	50		17.02	40	65%	10,000
7.04	4.21	28.62	34.65	34.23	50		10.66	48	69%	10,000
7.03	4.19	28.62	35.07	34.22	50		11.22	48	68%	10,000
6.62	4.18	13.82	14.35	13.96	50		8.81	48	39%	10,000
4.44	2.14	21.52	17.73	16.37	100		3.97	72	78%	10,000
4.36	2.14	15.33	14.60	13.19	100		4.49	68	69%	10,000
4.49	2.14	21.54	17.85	16.57	100		3.87	68	78%	10,000
2.86	1.36	13.88	13.75	10.73	150		3.95	75	71%	10,000
2.76	1.38	9.42	9.07	6.85	150		3.61	75	60%	10,000
2.86	1.37	9.78	9.63	6.65	150		3.30	75	66%	10,000
2.11	1.68	10.21	9.67	7.15	200		2.82	100	71%	10,000
4.02	1.69	8.90		12.66	50	10.50	8.03	45	17%	10,000
7.76	2.58	19.46		22.32	50	11.43	9.86	26	49%	10,000
2.25	1.20	9.59		8.67	100	4.78	3.41	86	45%	10,000
2.84	0.83	9.60		9.52	100	6.78	3.24	67	29%	10,000
20.33	8.06	98.63		82.94	100	39.56	35.59	91	52%	100,000

small, and communication between the master and workers, which is ignored in the model, has a secondary but non-negligible impact on total running time. Since practical problems of interest involve large planning problems, the model that ignores communication costs is used below to minimize total ATR execution time.

4.2 Adaptive Code for Homogeneous Workers

Based on the above results, we can specify an optimal adaptive configuration for ATR algorithm, for a given number of scenarios *N* and a number of available homogeneous workers, assuming the objective is to minimize the total ATR execution time. The model could be applied in a similar way to achieve some other objective, such as a balance between minimizing execution time and maximizing processor efficiencies, which is a subject left for future work.

To (nearly) minimize total execution time on homogeneous workers, *B* should be set to 1, the number of task groups *G* should be set equal to the number of workers, and tasks should be distributed equally among the workers. *T* could be specified by the user, or can be set by the adaptive code to 200 or 400, motivated by the fact that, for most planning problems and *T* in this range, total master execution time does not increase greatly, while the number of iterations decreases significantly. Table 3, as well as many other experiments omitted to conserve space, show that for various planning problems, number of scenarios, and number of workers, with *B*=1 and *G* equal to the number of workers, the running time decreases as we increase *T* in the range of 25-400. Due to diminishing returns in reducing the number of iterations, values of *T* larger than a few hundred, for the representative planning problems studied in this paper, do not improve total ATR running time. We note that if ATR is modified to allow *G* to exceed *T* (a change that is simple in principle), then ATR could still make productive use of more than 400 workers while still using a near-optimal value of 200 – 400 for *T*.

The optimal configuration for a fixed number of homogeneous workers is easily adapted to the case where the number of workers changes during the execution of the program. In this case, each worker currently available is given approximately the same number of tasks to evaluate, so that the time that the master needs to wait for results from the workers is minimized. In the current version of ATR, where each task is evaluated by a single worker, it is valuable for *T* to be large (e.g., 400) because the work can be distributed more evenly across the workers as the number of workers varies.

4.3 Heterogeneous Grids

Table 4 shows that, unless the user requests a homogeneous worker pool, the processors allocated to ATR by Condor may have very diverse speeds, typically differing by a factor of seven to ten. This is also true in other grid environments. Equal partitioning of the work between processors is not a particularly good strategy in this case, as the master processor will need to wait for the slowest worker to complete, leaving faster workers idle.

A better approach would be to give each worker a fraction of the total scenarios *N* proportional to its speed. For instance, if we are evaluating *N* scenarios, a worker with peak processing rate M_i MIPS would receive $(N/M) \times M_i$ scenarios to evaluate, where *M* is the total peak rate summed over all workers. However, the algorithm for computing the next iterate on the master requires *T* (i.e., the number of subgradients computed per iteration) to be fixed throughout the computation, and the work assigned to each worker must be an integral number of tasks of size *N*/*T*.

Table 5: Execution Time Comparisons with Previous ATR
(SSN, N=40,000, 50 Workers)

Worker Pool	Original ATR Recommended Values of B, G, and T Execution Time (T = 100, G = 25)				New Adaptive ATR Execution Time
	Reduced Debug		Default Debug		
	B=3	B=6	B=3	B=6	
Homogeneous	61 min	92 min	68 min	149 min	18 min
Heterogeneous	80 min	112 min			25 min

Thus, the near-optimal algorithm to minimize total execution time for a given collection of heterogeneous worker nodes is as follows. *T* (the number of tasks) is again chosen to be moderately large (e.g., 400-800), so as to create smaller tasks for balancing the load across the heterogeneous workers. Each task contains *N*/*T* scenarios. Using the benchmark results for each worker, tasks are allocated one at a time to workers, such that each task will have the earliest expected completion time given the task assignments made so far. In this way, tasks are assigned to a worker in proportion to its execution time for the benchmark, such that the number of assigned tasks multiplied by the benchmark time will be approximately the same for each worker that is assigned at least one task. Some of the workers with high benchmark times might not be assigned any tasks, while workers with low benchmark times may be assigned multiple tasks.

Since this task scheduling algorithm is not implemented in the current MW runtime library, we implemented it inside the ATR application to experiment with its effectiveness. We are also collaborating with MW developers who are implementing this feature within MW.

Since the tasks can be assigned to each worker as each new iterate is created, the schedule is adaptive in nature. It also has the advantage of simplicity. Although the number and computational speeds of the workers may change dynamically during the run, the adaptive code yields minimum execution time without taking a complex global view of the runtime environment.

Table 4 shows the predicted and measured results of applying this near-optimal approach. The first eleven rows show the predicted execution time of the adaptive code, for workers that were allocated to the non-adaptive version of ATR. For the highly heterogeneous allocations, the ATR runtime is reduced by a factor of greater than three when the scheduling strategy that adapts to the worker speeds is applied. The lower part of the Table shows measured and predicted execution times from the experimental implementation of the adaptive scheme, along with the predicted total execution time if these runs had been performed with the non-adaptive code. For these somewhat less heterogeneous processor pools, factor-of-two speedups are estimated for the adaptive code. More significant speedups can be anticipated when the number of allocated workers changes greatly during the ATR execution. Table 5 also shows that, compared with ATR execution times for parameter settings recommended in the previous "rules of thumb", the new adaptive ATR has speedups that are a factor of four to eight on homogeneous workers, or a factor of three to four on heterogeneous workers.

5. CONCLUSION

We have performed a detailed analysis of the execution of the ATR stochastic optimization code running in a Condor grid environment. Initial measurements of the application, in this work as well as in previous work, showed highly variable execution times for key components of the algorithm, particularly on the master processor. In previous work, this issue was addressed by creating more parallel tasks than the number of workers, so that workers could productively evaluate scenarios during the long and unpredictable master computations. However, a more detailed analysis revealed simple mechanisms for reducing the variability of the task execution times, as well as a more complete understanding of the complex impact of the configuration parameters on total ATR execution time. Using the analysis, we developed and applied surprisingly simple performance models to determine configurations of ATR that minimize total execution time on either static and dynamic collections of homogeneous or heterogeneous workers. Experiments in a local Condor pool, as well as with widely distributed Condor flocks, indicate that total execution time is reduced, using the simple model-based adaptive execution, by factors of four to eight compared with the non-adaptive execution and using previously recommended configuration parameters. In addition, the new adaptive ATR uses a task-scheduling algorithm that can improve the performance of other parallel grid applications. This algorithm is currently being implemented in the Condor-MW library. The temporarily isolated master is also a proposed improvement in the runtime environment that could greatly benefit other master worker grid applications.

Ongoing research includes (1) applying the ATR model to more complex objectives, such as those that take into account the utilization of allocated processors as well as the ATR execution time, (2) developing models to control the adaptive execution of other complex codes, using the same approach, which emphasizes simplicity as well as accuracy, as we've used for ATR, and (3) improvements in ATR such as new heuristics for updating the model function *m*(*x*) and assigning partial tasks to workers to achieve better load balancing and/or higher degrees of parallelism in evaluating a single iterate. Although the development time for a simple high fidelity analytic model is substantial, (a) it is still a very small fraction of the time to design and develop a complex code such as ATR that will potentially be used to solve many important problems, and (b) the payoffs from the model in optimizing the adaptive execution can be significant. We also surmise that the LogP class of models [1][7] is a reasonable starting point for developing other model-based adaptive codes, since previous models of simple adaptive applications (reviewed Section 2.3) as well as the simple model developed in this paper for ATR, can be viewed as LogP models, and since a LogGP model of a complex non-adaptive particle transport code [26] is also highly accurate.

ACKNOWLEDGEMENTS

The authors thank Jeff Linderoth for many helpful discussions concerning the ATR solution algorithms and for help in running the experiments on widely distributed Condor flocks, and Jichuan Chang for significant improvements to Condor-MW that enabled this work.

REFERENCES

[1] Alexandrov, A., M. Ionescu, K. E. Schauser, and C. Scheiman, "LogGP: Incorporating Long Messages into the LogP Model", *Proc. 7th Ann. ACM Symp. on Parallel Algorithms and Architectures,* Santa Barbara, CA, July 1995.

[2] Allen, G., W. Benger, T. Goodale, H. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel, and J. Shalf, "The Cactus Code: A Problem Solving Environment for the Grid", *Proceedings of the 9th IEEE Int'l. Symposium on High Performance Distributed Computing,* Pittsburgh. 2000.

[3] Berman, F. and R. Wolski, "Application –Level Scheduling on Distributed Heterogeneous Networks", *Supercomputing November, 1996*.

[4] Birge, J. R., C. J. Donohue, D. F. Holmes, and O. G. Svintsiski, "A parallel implementation of the nested decomposition algorithm for multistage stochastic linear programs," *Mathematical Programming* 75 (1996), pp.327-352.

[5] Birge, J. R. and L. Qi, "Computing block-angular Karmarkar projections with applications to stochastic programming," *Management Science* 34 (1988), pp. 1472-1479.

[6] Chang, F. and V. Karamcheti, "A Framework for Automatic Adaptation of Tunable Distributed Applications", *Cluster Computing: The Journal of Networks, Software and Applications, May,* 2001.

[7] Culler, D., R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. Von Eiken, "LogP: Towards a Realistic Model of Parallel Computation", *Proc. 4th ACM SIGPLAN Symp. On Principles and Practice of Parallel Programming (PPoPP '93),* San Diego, CA, May 1993.

[8] Dail, H., G. Obertelli, F. Berman, R. Wolski, A. Grimshaw, "Application-Aware Scheduling of a Magnetohydrodynamics Application in the Legion Metasystem", *Proc. 9th Heterogeneous Computing Workshop,* May 2000.

[9] Epema, D. H. J., M. Livny, R. van Dantzig, X. Evers, and J. Pruyne, "A worldwide flock of Condors: Load sharing among workstation clusters," *Journal on Future Generation Computer Systems 12* (1996), pp. 67-85.

[10] Foster, I. and C. Kesselman, "The Globus Project: A Status Report," in *Proceedings of the Heterogeneous Computing Workshop*, IEEE Press, 1998, pp. 4-18.

[11] Fragniere, E., J. Gondzio, and J.-P. Vial, "Building and Solving Large-Scale Stochastic Programs on an Affordable Distributed Computing System," *Annals of Operations Research* (2000).

[12] Goux, J.-P., S. Kulkarni, J. Linderoth, and M. Yoder, "An enabling framework for master-worker applications on the computational grid," *Proceedings of the IEEE Symposium on High-Performance Distributed Computing*, 2000.

[13] Grimshaw, A. S., Wm. A. Wulf, and the Legion Team, "Legion: The next logical step toward the world-wide virtual computer," Communications of the ACM 40 (1997).

[14] Heymann, E., M. Senar, E. Luque, and M. Livny, "Adaptive Scheduling for Master-Worker Applications on the Computational Grid", *GRID 2000*.

[15] Kapadia, N. H., J. A. B. Fortes, and C. E. Brodley, "Predictive Application-Performance Modeling in a Computational Grid Environment", *8th IEEE Int'l. Symposium on High Performance Distributed Computing,* August 1999.

[16] Keleher, P. J., J. K. Hollingsworth, and D. Perkovic, "Exposing Application Alternatives", *19th Int'l Conference on Distributed Systems, 384-392*, June 1999.

[17] Lan, Z., V. E. Taylor, "Dynamic Load Balancing of SAMR Applications on Distributed Systems", *Proc. Supercomputing 2001*.

[18] Linderoth, J. and Wright, S. J., "Decomposition Algorithms for Stochastic Programming on a Computational Grid," Preprint ANL/MCS-P875-0401, MCS Division, Argonne National Laboratory, April, 2001., University of Wisconsin-Madison, October, 2001.

[19] Livny, M., J. Basney, R. Raman, and T. Tannenbaum, "Mechanisms for High-Throughput Computing," *SPEEDUP* 11 (1997).

[20] Ribler, R. L., H. Simitci, D. A. Reed, "The Autopilot Performance-Directed Adaptive Control System", *Future Generation Computer Systems*, special issue (Performance Data Mining), Volume 18, Number 1, 2001. pp. 175-187.

[21] Ripeanu, M., A. Iamnitchi, "Cactus Application: Performance Predictions in a Grid Environment", *Conference on Parallel Computing*, 2001.

[22] Salmon J.J., Warren M.S., Winckelmans G.S., Fast Parallel Tree Codes For Gravitational And Fluid Dynamical N-Body Problems, *Int J Supercomputing Apps.* 8: (2), pp. 129-142, 1994.

[23] Smallen, S., H. Casanova, F. Berman, "Applying Scheduling and Tuning to On-line Parallel Tomography", *Supercomputing 2001*.

[24] Spring, N. and R. Wolski, "Application Level Scheduling of Gene Sequence Comparison on Metacomputers", *Proc. 12th ACM Int'l. Conf. on Supercomputing*, July 1998.

[25] Su, A., F. Berman, R. Wolski, and M. M. Strout, "Using AppLeS to Schedule Simple SARA on the Computational Grid", *UCSD Tech Report,* *http://apples.ucsd.edu/netpubs.html*, February 1999.

[26] Sundaram-Stukel, D., and M. Vernon, "Predictive Analysis of a Wavefront Application Using LogGP", *Proceedings of 7th ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming,* May 1999.

[27] Vraalsen, F., R. A. Aydt, C. L. Mendes, and D. A. Read, "Performance Contracts: Predicting and Monitoring Grid Application Behavior", *Proceedings of the 2nd Int'l. Workshop on Grid Computing,* November 2001.