

Deployment of Enterprise-Scale Systems Sustainability Optimization on Commodity Computing Clusters

Jean-Paul Watson • David R. Strip • David L. Woodruff

*Sandia National Laboratories
Discrete Math and Complex Systems Department
Albuquerque, NM 87185, USA
jwatson@sandia.gov • drstrip@sandia.gov*

*Graduate School of Management
University of California, Davis
Davis, CA 95616, USA
dlwoodruff@ucdavis.edu*

Enterprise-scale military logistics operations for system sustainability are now commonly modeled as large-scale stochastic simulation models, as opposed to more traditional analytic models. While typically possessing less restrictive and unrealistic modeling assumptions, optimization wrappers for these simulation models are significantly more difficult to solve than simpler analytic models. In particular, the optimization models are naturally expressed as stochastic mixed-integer programs, which are notoriously difficult, especially relative to their deterministic, mixed-integer programming counterparts. Consequently, deployment to real-world customers is a significant issue. We describe our experience in developing a practical, stochastic programming solution for efficiently optimizing the enterprise-scale logistics networks for sustaining the operation of the Lockheed Martin Joint Strike Fighter and the US Army Future Combat Systems. Our approach, based Rockafellar and Wets' Progressive Hedging (PH) algorithm, is naturally and efficiently parallelized on the small-to-moderate-scale computing clusters that end-users may possess, and can effectively solve stochastic mixed-integer programs with hundreds to thousands of decision variables. Unfortunately, the acquisition and maintenance costs of computing clusters prevent many end-users from accessing and leveraging parallel compute resources. To bridge this gap, we discuss deployment of our parallel PH algorithm on the Amazon EC2 compute cluster. Finally, we detail various real-world requirements driving the deployment of our parallel PH solution. (*Stochastic Programming, Progressive Hedging, Logistics, Parallel Computing, Supply Chain Optimization, Systems Sustainability*)

1. Introduction

Acquisition and initial deployment costs of military systems are significant, as any casual reader of the popular press will observe. Less appreciated is the fact that the cost required to sustain these systems through years or decades of operational life – whether peacetime or wartime – generally far exceeds these “up-front” costs. The same is true for commercial systems such as oil rigs, airline fleets, and server-class computer systems. For any of these systems, a primary concern during pre-deployment planning is to provide sufficient logistical support in the form of spare parts and repair-related resources to ensure that the deployed systems remain operational with sufficiently high probability, at the minimal possible cost. A notional example involves the deployment of various squadrons of fighter aircraft at bases throughout the world. A typical optimization objective in this context is to minimize the investment cost in the logistics network while ensuring that each aircraft in each squadron is available to fly missions 90% of the time. The logistical system in these problems consists of original equipment manufacturers (OEMs) to build, stock, and repair parts, supply depots to store spare parts, repair depots for fixing high-cost “recoverable” parts such as engines, and bases with limited supply and repair functionality; the various sites are typically arranged in a complex, multi-echelon structure. The decision variables in such an optimization problem include, for each site in the logistics system: the quantity of each repair-related resources, the initial allocation of spare parts, and the parameters associated with part inventory re-order policies. We refer to this general class of logistics network optimization problem as the Support Enterprise Sustainability Problem, or SESP. For a more detailed overview of this problem class, we refer the reader to Muckstadt (2005).

Beginning in the 1960s, SESP were formulated and solved using analytic optimization models such as METRIC (Sherbrooke, 1968), VARI-METRIC (Slay, 1984), and most recently, ASM (Slay and King, 1987). These models make certain assumptions regarding the distribution of part failure times on deployed systems, logistics network structure, repair turn-around times, and inter-site transportation times. Given these assumptions and target performance (i.e., availability) levels for a deployed system, these models can efficiently locate minimal-cost solutions. As a consequence, these models have seen widespread adoption throughout all branches of the military and in various commercial contexts.

Despite the success of these analytic models, there has recently been an emerging and accelerating shift toward explicit simulation (e.g, discrete event) models of the SESP. Prominent examples include the Support Enterprise Model (SEM) (Smith et al., 2006), which is used by Lockheed Martin to analyze the SESP for the F-35 Joint Strike Fighter, and the System-of-Systems Analysis Toolkit (SoSAT) for analyzing the SESP for the US Army’s Future Combat Systems. These simulations are being used in the planning / pre-production phases for the corresponding systems, in order to assess the impact of system reliability and logistics network changes on life-cycle sustainability costs. In the case of Lockheed Martin, the primary objective is to develop a low-cost, efficient logistics network in order to secure contracts for F-35 sustainability. In the case of the US Army, the objectives are to minimize long-term support costs for FCS and to identify design improvements in the logistics networks to improve system availability.

The shift from analytic models to simulation models is driven by three primary factors: 1) customers are viewing the assumptions underlying models such as METRIC and VARI-METRIC as increasingly restrictive, 2) analytic models generally fail to provide a mechanism to accurately capture and quantify variability in a logistics system, and 3) customers wish to assess the impact of imposing various “business rules” (e.g., priority rules for deciding which bases should get a spare part at a particular point in time) on logistics network performance. Simulation models naturally address these shortcomings, but incur two significant costs. First, accurate simulation of a large-scale SESP with global deployment scope, hundreds of sites, thousands of deployed systems, each with hundreds to thousands of parts, incurs non-trivial run-times for individual simulation replications. Second, existing analytic models are no longer applicable due to variable levels of departure from the underlying set of assumptions under which they were originally developed.

We have begun to address these two issues by developing a novel approach to optimizing the SESP *given* an underlying simulation of a logistics system sustainability network (Watson et al., Submitted). We formulate the SESP as a stochastic mixed-integer program (Kall and Wallace, 1994), which we then solve using Rockafellar and Wets’ Progressive Hedging (PH) scenario-based decomposition algorithm (Rockafellar and Wets, 1991). By leveraging various computationally and mathematically motivated algorithmic techniques, we are able to solve individual instances of very large-scale SESPs (e.g., for the Joint Strike Fighter) in hours to tens of hours of computing time, each possessing thousands of decision variables. Although this represents a significant advance in terms of optimization technology for simulation-

based SESP, a number of serious practical barriers related to end-user deployment remain, all driven by the need to significantly reduce run-times in order to increase analysis turn-around times.

In this paper, we detail our efforts to address these deployment barriers through parallelization of our PH approach for solving the SESP on widely available “commodity” parallel computing environments, i.e., Beowulf clusters. We begin in Section 2 with an overview of both the target application – optimization of enterprise-scale SESP – and the formulation of SESP as stochastic mixed-integer programs. The PH algorithm for solving SESP is then introduced in Section 3. Although effective for solutions of SESP, a number of real-world factors drive the need for significant speedup, through parallelization, of the basic serial PH algorithm. We discuss these factors in Section 4. Our approach to parallelization of PH is discussed in Section 5. For purposes of data sensitivity, we cannot disclose results on the Joint Strike Fighter SESP or in-progress work on the Future Combat Systems SESP. Consequently, we developed synthetic, publically available data sets for dissemination of results; these data sets are described in Section 6. Performance results of our parallel PH algorithm on both a typical Beowulf cluster and Amazon’s EC2 cluster are respectively discussed in Sections 7 and 8. We conclude with a discussion of the implications of our results in Section 9.

2. The SESP: Description and Formulation

A detailed description of the SESP is provided in (Watson et al., Submitted); in this section, we provide a brief overview with an emphasis on the problem-solving approach. In a given SESP problem instance, the decision variables consist of (1) stock and re-order levels for each part at each site in the system; relevant sites include all supply depots (independent of echelon level), OEMs, and bases, and (2) assigned quantities for all repair-related resources for each site in the system; relevant sites include repair depots, OEMs, and bases. We assume a (s, q) inventory re-order policy, $s > q$, with the initial procurement level for each part/site combination set to some n , $s \leq n \leq q$. The cost of an SESP solution is given by the procurement cost of the initial spare parts and resource allocations, in addition to any ongoing operational and maintenance costs associated with resources. Various secondary costs, including those associated with the replacement of consumable parts, site-to-site transportation, and shop materials for part repair, are generally ignored; these costs are “sunk” in the

sense that they are in practice unavoidable once the spares and resource levels throughout a logistics system are determined. The optimization objective is to identify a SESP solution that minimizes cost while ensuring that target percentage availability statistics for deployed systems are achieved. Availability of a system is locally dictated by the presence of spare parts to replace failed components; at the global level, additional factors include the ability of supply depots to ship parts, the capacity of repair depots to fix failed parts, and the lead time required to procure new replacement parts.

As discussed in Section 1, it is becoming commonplace for customers to model SESP through the use of simulation tools, due largely due to the complexity of operational failure distributions (e.g., in the case of combat) and imposition of customer-specific business rules. Two examples of simulation models we are currently involved with include the Support Enterprise Model (SEM) (Smith et al., 2006) and the System of Systems Analysis Toolkit (SoSAT), both developed at Sandia National Laboratories. SEM is targeted toward SESP involving military aircraft, while SoSAT is designed to model ground combat systems such as a tank brigade. SEM and SoSAT are respectively in active use for analysis of the Lockheed Martin Joint Strike Fighter (JSF) and US Army Future Combat Systems (FCS) SESP. While powerful, the flexibility and resolution of these simulations comes with a price: evaluation of individual solutions, e.g, to determine system availabilities, is generally expensive, requiring minutes to hours of run-time for individual simulation replications. Given thousands to tens of thousands of decision variables, traditional “black-box” optimization approaches (such as simulation-based optimization (Gosavi, 2004)) are therefore not feasible in this context.

To develop our optimization “wrapper” for SESP simulation models, we leverage the underlying simulation in a limited role, specifically to provide *input* to the optimization process. Given an SESP instance, the corresponding simulation model is executed to generate part failure data for each system, for some number of independent replications or scenarios. These replications are executed in a “flooded” mode, i.e., one in which the supply of parts and available resources is unconstrained. By directly leveraging the simulation model in this fashion, it is possible to sample part failures from non-parametric or not generally accessible (from an analytic standpoint) parametric distributions, e.g., complex wear-out distributions or failure due to combat damage. The resulting part failure sequences are necessarily optimistic relative to a cost-constrained environment. In particular, part failures are assumed to be independent. For example, consider a part failure sequence for a particular

plane from a flooded SEM replication in which a landing gear component fails on day n and the engine fails on day $n + 5$. In a parts-rich environment, the landing gear is quickly repaired, such that the engine will fail due to the aircraft being operational; in a resource-constrained environment, lack of a spare landing gear component may down the plane for $n > 5$ days, in which case the engine failure would be delayed. However, given the typically high availability requirements (on the order of 90% through 95%) for nearly all deployed military and commercial systems, the degree of conservatism is in practice not significant.

For any *given* SESP scenario, $s \in \mathcal{S}$, it is conceptually straightforward to develop a mixed-integer programming (MIP) formulation to express the cost minimization of the support enterprise, subject to the constraint that average availability of each system is greater than or equal to a user-specified threshold. Such a MIP must track state variables such as on-hand and due-out inventory quantities, repair queue and repair in-process status, and the number of systems downed due to lack of a spare part. Constraints in the MIP then conserve inventory positions across time, enforce limits on the utilization of repair resources, and model inter-site transport delays; there are significant nuances related to the corresponding MIP model, which are detailed in Watson et al. (Submitted) and Greenberg (2007). Given $|\mathcal{S}|$ scenarios, each with distinct part failure time-series, we define a stochastic MIP for the *aggregate* SESP through linking constraints, simply enforcing that the value of any given decision variable is identical in all individual scenario MIP solutions. The aggregate SMIP model is known as an *extensive form* formulation of the SESP.

Ideally, it would be possible to solve the extensive form of the SESP directly with existing commercial MIP solvers such as CPLEX (ILOG). However, this is currently not computationally feasible due to both the size and empirical difficulty of the resulting problems; even solution of MIPs corresponding to individual scenarios is too expensive given large-scale, real-world SESP instances. In response, we have developed powerful domain-specific heuristics for solving the MIPs corresponding to individual SESP scenarios, achieving solutions in seconds to minutes of run-time. The difficulty of the MIP formulation and details regarding the heuristics are described in Watson et al. (Submitted). Given high-quality solutions for individual SESP scenarios, the question is then: How can solutions to individual scenarios be aggregated to form a single, low-cost solution to the extensive form of the SMIP that simultaneously satisfies performance constraints in all scenarios.

3. An Overview of Progressive Hedging

Numerous algorithms for solving stochastic mixed-integer programs (SMIPs) (Kall and Wallace, 1994) are based on decomposition by time stages of the scenario tree, i.e., a tree in which vertices represent decision points, edges represent specific decisions, and the leaves represent the set of all possible outcomes. Alternatively, horizontal decomposition algorithms decompose SMIPs by complete scenarios, i.e., via specific and complete paths through the scenario tree. Progressive Hedging (PH), introduced by Rockafellar and Wets (1991), is a horizontal decomposition approach to solving SMIPs. PH is particularly appropriate when there exist good, fast heuristics for generating solutions to individual scenarios. As discussed above in Section 2, this is the case for our formulation of the SESP.

For an individual scenario s , many problems of practical interest can be cast in the general framework of constrained optimization:

$$\begin{aligned} & \text{Minimize} && c \cdot x_s && (P_s) \\ & \text{Subject to:} && x_s \in \mathcal{Q}_s \end{aligned}$$

where x_s is a decision vector of length n_s , c is a vector of cost coefficients, and the requirement $x_s \in \mathcal{Q}_s$ expresses the problem constraints, i.e., to ensure x_s is a feasible solution. We use the subscript s to emphasize that the specific problem characteristics will depend on the scenario that is actually observed.

For each scenario $s \in \mathcal{S}$, we denote the probability of occurrence by $\Pr(s)$. These probabilities allow us to take into account prior knowledge of the distribution of individual scenarios, or to weight the relative importance of particular scenarios based on problem-specific knowledge. For the operational decisions associated with the SESP, the goal is to minimize expected investment cost, which can be written as:

$$\begin{aligned} & \text{Minimize} && \sum_{s \in \mathcal{S}} \Pr(s)(c \cdot x) && (\text{EF}) \\ & \text{Subject to:} && x \in \mathcal{Q}_s \end{aligned}$$

where the use of the decision vector x ($x_s = x, \forall s \in \mathcal{S}$) that does not depend on the scenario implicitly implements the *non-anticipativity* constraints that avoid allowing the decisions to depend on a particular scenario.

For the optimization problem EF, the basic PH algorithm can be stated as follows, taking a perturbation factor $\rho > 0$ as the sole input parameter:

1. $k := 0$

2. For all scenarios $s \in \mathcal{S}$

$$x_s^{(0)} := \underset{x}{\operatorname{argmin}}(c \cdot x) : x \in \mathcal{Q}_s$$

3. $\bar{x}^{(0)} := \sum_{s \in \mathcal{S}} \operatorname{Pr}(s) x_s^{(0)}$

4. $w_s^{(0)} := \rho(x_s^{(0)} - \bar{x}^{(0)})$

5. $k := k + 1$

6. For all scenarios $s \in \mathcal{S}$

$$\begin{aligned} x_s^k &:= \underset{x}{\operatorname{argmin}}(c \cdot x \\ &+ w_s^{(k-1)} \cdot x + \rho/2 \|x - \bar{x}^{(k-1)}\|^2) \\ &: x \in \mathcal{Q}_s \end{aligned}$$

and

$$w_s^{(k)} := w_s^{(k-1)} + \rho(x_s^{(k-1)} - \bar{x}^{(k-1)})$$

7. $\bar{x}^{(k)} := \sum_{s \in \mathcal{S}} \operatorname{Pr}(s) x_s^{(k)}$

8. If the termination criteria are not met, then go to step 5.

The termination criteria are based mainly on the convergence of the $x_s^{(k)}$ to a common \bar{x} , although non-convergence in the case of non-convex optimization problems such as the SESP is a possibility, so termination may also be based on an iteration limit.

Integer constraints on elements of the decision vector x render stochastic programming problems non-convex and significantly increase the difficulty of solution. A variety of algorithms for solving the resulting SMIPs have been proposed (e.g., see (Maarten and van der Vlerk, 1996–2003)). For some smaller problem instances, standard mixed-integer programming (MIP) solvers can be used (Parija et al., 2004) to directly solve the extensive form EF of the problem. However, standard MIP solvers fail to consistently solve even individual scenario sub-problems in for our SESP formulation, let alone the extensive form of the problem (Watson et al., Submitted).

In contrast, PH is a natural algorithm for solving large-scale SMIPs via scenario decomposition. Although the integer variables also add complexity to solution via the PH algorithm, they can be used to speed convergence because equality is well-defined and easily detected (Løkketangen and Woodruff, 1996). An alternative approach is to use PH to solve a variant

of the SESP where the integer restrictions are relaxed and then round the solution obtained at convergence (Listes and Dekker, 2005). Unfortunately, this technique yields poor-quality solutions in the case of the SESP.

The intent of this section is to provide a concise overview of the PH algorithm, with detail sufficient to understand the issues involved in parallelization of PH, as described subsequently in Section 5. Further detail on the solution of SMIPs can be found in Berland and Haugen (1996) and Mulvey and Vladimirou (1991), while examples of specific applications of PH are reported in Listes and Dekker (2005) and Løkketangen and Woodruff (1996). To achieve convergence in tractable run-times for the SESP in the serial case, we introduced a number of computationally and mathematically motivated techniques that significantly complicate the core PH algorithm. However, none of these modifications impact the nature of the PH parallelization. These techniques, in addition to further modeling considerations encountered in applying PH to the SESP, are fully detailed in Watson et al. (Submitted).

4. Parallel Solution of the SESP: Requirements and Impact

Prior to considering parallelization of our PH algorithm, we first address the obvious question: What end-user requirements are driving the need to develop a parallel implementation of PH for the SESP? As discussed below in Section 7, and in our prior empirical analysis (Watson et al., Submitted), the serial run-times of PH on small-to-medium sized SESP instances range from hours to days on modern workstations. Fundamentally, the SESP is a strategic planning problem; response times of 1-2 days would perhaps not appear to be a deployment issue. However, as we now discuss, three real-world factors drive the ultimate need for faster solution times:

Significant Expected Growth in the Number of Scenarios: For the SESP corresponding to the JSF logistics system, we consider at most between 30 and 60 scenarios. These small scenario counts are driven by the duration of the planning horizon, which is typically at most several years. With such long horizons, relatively few scenarios are required to observe a full range of system responses, as system performance metrics are quantities averaged over the entire duration of the simulation. In contrast, our work on the FCS logistics system indicates that several hundred scenarios are required to achieve the same degree of coverage; the duration of the FCS planning horizon is

on the order of days to weeks. We expected other SESP, especially in the commercial domain, exhibit a similar characteristic. Larger numbers of scenarios significantly increases the run-time time of PH, due to both increases in the cost of solving individual scenarios and in the aggregate number of PH Iterations. Further, even in the case of JSF, consideration of larger numbers of scenarios yields improved confidence in the ability of solution performance to generalize to unobserved scenarios.

Design is an Inherently Iterative Process: Perhaps counter to intuition, real-world strategic planning problems such as the SESP are solved numerous times. Simulation input parameters are refined over time, as the characteristics of system components become better understood, and with higher degrees of accuracy. Optimization objectives and side-constraints are frequently redefined following analysis of prior optimization runs that identify possible trade-offs and binding constraints in system performance metrics. In the cases of parameter and objective refinement, solution times of approximately ten hours are a realistic performance target for our PH algorithm, as this allows execution of optimization runs overnight. During the course of a work day, analysts can then refine input simulation databases, analyze optimization results from the previous night’s run, or specify new optimization targets and side constraints – all in preparation for a new overnight optimization run; the process then repeats.

Analysts Generate Ideas Faster Than They Can Be Assessed: Analysts commonly express the desire to expend additional computational cycles – assuming they are available – investigating “what-if” questions, to ensure a more broad exploration of the system design space. For any reasonably complex SESP, there universally exist more hypothetical optimization scenarios to assess than could ever be practically computed. However, the importance of these analyses should be not be under-stated, as they serve to significant increase end-user confidence in the quality and performance of an SESP solution. In terms of run-time targets, “as fast as possible” is the only constraint we have in practice to satisfy this requirement, at least while PH solve times remain in the range of tens of minutes to days.

5. Parallelization of Progressive Hedging

As is evident in the pseudo-code presented in Section 3, each iteration of the PH algorithm consists of a set of independent optimizations for individual scenarios $s \in \mathcal{S}$ followed by computation of the weights $w_s^{(k)}$ and averages $\bar{x}^{(k)}$ for use in the next iteration of the algorithm. In the SESP, as is typical in most applications of PH, the bulk of the computation is in the optimization for the individual scenarios, with only minor computation required to calculate the weights and averages. While the scenario solves are independent and can be easily parallelized, the PH algorithm must be synchronized prior to execution of the subsequent PH iteration, as both individual scenario solves and weight adjustments rely on the updated $\bar{x}^{(k)}$. Suppose that the individual scenario solve times are roughly equal, with individual processors (assuming one scenario is assigned to each processor) spending little time waiting for the slowest solve to complete. In the supercomputing community, this sort of situation is referred to as “embarrassingly parallel” – a problem in which the ratio of computation to communication is very high, processors tend to spend very little time waiting; this paradigm thus has the potential to scale efficiently on parallel computers. The question of whether individual scenario solve times are roughly equal (a necessary condition to achieve large speedups in parallel computing environments) is an empirical one, and is addressed in Sections 7 and 8.

The relatively high computation-to-communication ratio exhibited by PH on the SESP allows for potentially efficient parallelization on even the most loosely coupled computing clusters. To maximize deployment flexibility, we implemented our parallelization strategy using remote procedure calls (RPC), rather than a more sophisticated (and restrictive) approach such as MPI, which significantly increases the level of technical expertise required to assemble and manage a compute cluster. RPC provides a set of standards that allow a program running on one compute node to pass data and invoke subroutines on another compute node via a network connection such as TCP/IP. A brief introduction can be found online (RPC Overview, 1997). The RPC standard is also available online (RPC Specification, 1988). Bloomer (1992) provides a detailed exposition of RPC and various examples. RPC is a simple instance of the client-server communication paradigm. The server runs continuously, waiting for a client to request a service. The RPC mechanism accepts the incoming message from the client, unpacks the data into the machine’s native format from the network neutral transmission format (XDR), and dispatches the call to the appropriate

subroutine. This mechanism allows for deployment on a heterogeneous set of computing nodes. The machinery behind this processing is largely hidden from the developer, who can use the RPCGEN (Unix) or MIDL (Windows) preprocessors to translate an interface specification into the necessary code.

In the case of our implementation for the SESP, the server code has three functions exposed via RPC: optimize a scenario with no consideration of weights (step 2 of the PH pseudocode), optimize a scenario accounting for variable weights (step 6 of the PH pseudocode), and terminate the server. The server is launched with a fairly simple command line that specifies the TCP port to use for the RPC connection and the prefix that is used to determine the locations of the various input and output files. By providing the RPC port assignment on launch rather than hardcoding a choice, we can manage multiple instances of the server on multi-core and multi-chip processors. On launch the SESP problem instance is read, defining the logistics system, the decision variables, all of the available scenarios, and ρ parameter value(s); a server can potentially generate solutions for any given scenario. The server then waits for a service request to arrive via RPC. Both the non-weighted and weighted RPC calls simply pass a scenario number as an argument, indicating which sub-problem is to be solved. In the case of the non-weighted solve, no additional input is required. In the weighted case, the client transmits files specifying the $w_s^{(k-1)}$ and $\bar{x}^{(k-1)}$ for a specific scenario $s \in \mathcal{S}$ prior to invoking the RPC call.

Upon completion of a scenario solve, the RPC call returns to the client, which then downloads a file specifying the resulting value for the decision vector x_s . While it is possible to use RPC to transmit data between the client and server, for simplicity we use secure copy (scp) or remote copy (rcp), depending on the specific environment. As discussed below in Section 7, communication times for this data are not a significant issue. In compute clusters where all nodes share a common file system, the cost of the explicit file transfers can be avoided, at least in principle. For example, a common cluster configuration possesses a large disk array on a head node, which is mounted via NFS on each compute node. Unfortunately, NFS may cache portions of the file system, making it difficult to determine if a particular input or output file is associated with the current or a previous iteration of PH. In our experiments, we experienced considerable problems with PH convergence specifically because we were often working with “stale” data.

The client, which is the external interface to the system, is written in two parts: an AMPL (AMPL, 2007) script that implements the PH algorithm and a dispatcher (written in C++)

that assigns scenarios to servers for solution during each iteration of the PH algorithm. The dispatcher is launched with a list of IP addresses and port numbers for the available servers, which may be smaller than the number of scenarios. At each iteration of the PH algorithm, the dispatcher works through the list of active servers, assigning unsolved scenarios to free servers as they become available. All file transfers are initiated by the dispatcher. In all of the parallel computational experiments reported in Sections 7 and 8, the client codes are executed on a compute node distinct from the servers. One advantage of this deployment configuration is that “head” compute node of a cluster is often multi-CPU or multi-core, allowing the dispatcher code to execute multiple parallel threads for RPC invocation and file transfer.

Clearly, the client is a potential bottleneck in the execution of PH, though in general the impact is quite small because it represents such a small fraction of the overall computation. Due to convergence accelerators we developed for PH (Watson et al., Submitted), individual scenario solve times are inversely proportional the PH iteration count k . Consequently, in the final PH iterations, the communication costs associated with RPC invocation and file transfers can overwhelm the efficiency of parallelizing the individual scenario solves. When this occurs (easily detected by straightforward heuristics), we terminate parallel execution of the dispatcher and instead solve all scenarios on the client compute node.

We conclude by observing that our approach to parallelizing PH is clearly straightforward. This choice was due simply to the near-linear speedups observed in the course of our computational tests (as described in Sections 7 and 8); alternative approaches may only achieve mild improvements in efficiency. Some of these approaches, including asynchronous versions of PH, are described in Somervell (1998).

6. The Benchmark Problems

In Sections 7 and 8, we empirically investigate the performance of parallel PH for solving SESP. The experiments are performed using a set of synthetic SESP instances, generated specifically for dissemination of results, as the full-scale SESP corresponding to both the JSF and FCS logistics enterprises are proprietary. The instances are based on a simple echelon network structure consisting of a single repair depot, supply depot, and OEM, in addition to n operational aircraft bases. Five aircraft, composed in a single squadron, are assigned to each of the n bases. Each squadron flies a single sortie consisting of two aircraft – assuming

two such aircraft are available – for 4 hours every day. Each aircraft consists of 50 modeled parts, representing a range of failure distributions (e.g., random and wearout) experienced during operational flying time. The part failure sequences for these instances were generated using the SEM simulator, and each instance contains failure data for $|\mathcal{S}|$ realizations or operational scenarios. The decision variables are restricted to the procurement levels for the spare parts at each site in the system. Repair depots are assumed to be uncapacitated, performing repairs in a fixed duration; this assumption mirrors that present in the METRIC (Sherbrooke, 1968) and other analytic SESP optimization models. Further details regarding the structure of the test problems (in addition to instances involving resource-related decision variables) are available in Watson et al. (Submitted).

We consider test problems with both $|\mathcal{S}| = 10$ and $|\mathcal{S}| = 30$ scenarios, in addition to $n = 2$, $n = 5$, and $n = 10$ bases, yielding a total of six instances. We note that solutions obtained with $|\mathcal{S}| > 30$ scenarios are not significantly different than for $|\mathcal{S}| = 30$ scenarios, i.e., $|\mathcal{S}| = 30$ is sufficient for these instances (due to the long time horizon and heavy operational pace) to achieve target performance on unobserved scenarios. The largest test problems we consider possess 528 decision variables, which is significant relative to most stochastic mixed-integer program benchmarks. All of the test problems are freely available for general use, and can be obtained by contacting the authors. Finally, we note that we have executed both serial and parallel versions of PH on significantly larger, real-world SESP – specifically those corresponding to the full JSF logistics network and scaled versions of the FCS logistics network. These differences in scale should be considered when interpreting the run-times of PH reported below.

7. Experimental Results: A Native Beowulf Cluster

We first examine the performance of our PH algorithm executing on a typical Beowulf cluster (Beowulf Overview, 2007), i.e., a simple parallel computing environment constructed using commodity processors and interconnect, as opposed to the more specialized, custom hardware historically associated with parallel computers. Beowulf clusters are representative of the parallel computing environments that typical customers, e.g., governmental agencies, defense contractors, and commercial firms, may possess. This includes the US Army and Lockheed Martin end-users for which our parallel deployment of PH is targeted. Our test cluster consists of a head node running dual Intel Xeon 2.8 GHz processors with 2GB of

RAM, and 42 Intel Pentium 4 2.4GHz compute nodes, each with 1GB of RAM. Because the memory footprint of our heuristic for solving individual scenarios is less than 5 Mb for the largest test problems we consider in this paper, the relatively low memory capacity per compute node is not a factor. The compute nodes are connected to the head node via standard 100MBs network interconnect. It is important to observe that although many researchers – particularly in Computer Science – view such a cluster as simplistic, many customers still consider deployment and support of Beowulf clusters to be a major challenge. As a consequence, Beowulf clusters are not as pervasive in customer environments as one might expect.

For each of our test problems, we allocate one compute node for each scenario. As discussed in Section 6, the time horizons for these test problems is sufficiently long – one year – such that numbers of scenarios $|\mathcal{S}| > 30$ yield no significant improvement in the ability of solutions to achieve performance targets in novel scenarios not considered by the PH algorithm. For each test instance, we consider two ρ parameter selection rules, f_{100K} and sep , both detailed in Watson et al. (Submitted). The f_{100K} rule generally yields lower-quality solutions than the $\kappa = sep$ rule, although the run-times are much lower. While not discussed in this paper, for purposes of replicability we document the following values for the remaining PH parameters (see Watson et al. (Submitted) for details): $\mu = 0$, $\lambda_q = 0.5\%$, and $\lambda_t = 0.5\%$. For each test problem, we execute five independent trials of our parallel PH algorithm, recording the elapsed (i.e., wall clock) time per trial. The baseline serial performance is assessed using a single trial executed on an arbitrary compute node, again recording the elapsed time. Multiple trials are executed in the parallel case due to variance in the file transfer and RPC communication times, which is often significant. The individual scenario solves are deterministic, with a single serial run being sufficient to establish a baseline.

Num. Bases (n)	Num. Scenarios	Serial Run-Time	Parallel Speedup on $ \mathcal{S} $ Compute Nodes
2	10	2.39	2.91 – 4.05
	30	20.57	6.27 – 7.70
5	10	11.94	5.52 – 7.86
	30	114.07	12.14 – 20.89
10	10	88.64	8.43 – 8.75
	30	681.06	25.06 – 26.17

Table 1: Speedup statistics for parallel PH on a standard Beowulf computing cluster, using the ρ selection strategy f_{100K} . Serial run-times are reported in minutes.

Num. Bases (n)	Num. Scenarios	Serial Run-Time	Parallel Speedup on $ \mathcal{S} $ Compute Nodes
2	10	40.15	3.31 – 3.69
	30	135.37	11.17 – 12.45
5	10	64.33	7.72 – 8.1
	30	668.99	22.03 – 23.77
10	10	566.08	8.13 – 8.22
	30	5334.44	24.64 – 25.48

Table 2: Speedup statistics for parallel PH on a standard Beowulf computing cluster, using the ρ selection strategy *sep*. Serial run-times are reported in minutes.

The experimental results obtained for the ρ selection rules f_{100K} and *sep* are respectively shown in Tables 1 and 2. The columns labeled “Serial Run-Time” record the baseline elapsed time to solve the test problem on a single compute node; units are in minutes. The last two columns record the range of speedups over the five trials of the parallel PH implementation relative to the serial PH baseline baseline, given the respective ρ selection rule. Speedup is given as the serial elapsed time divided by the parallel elapsed time; for $|\mathcal{S}| = 10$ and $|\mathcal{S} = 30|$, the theoretical maximums are respectively 10 and 30.

Due primarily to network latency variability, we observe considerable variability in speedups for a given test problem, particularly in the case of the ρ selection rule f_{100K} . The speedup distributions are Gaussian-like, with a tendency toward the center of the given interval. The variability decreases with increases in problem size, which is the expected behavior as the ratio of computation to communication time increases; individual scenario solve times are proportional to instance size. For the largest test problem ($n = 10$, $|\mathcal{S}| = 30$), the variability is minimal and would likely not be noticed by or impact an end-user. Speedups are slightly greater for the *sep* rule in the case of the smaller $n = 2$ and $n = 5$ test instances, as the increased number of total PH iterations results in an improved computation-to-communication ratio. For the larger $n = 10$ instances, this ratio is already large, yielding no significant difference in speedup under the two ρ selection rules.

In terms of absolute performance, the speedups observed on smaller problem instances are poor. For the two $n = 2$ instances, the allocated compute nodes are in aggregate idle at least half the time. Improved speedups are observed for the two $n = 5$ instances, where efficiency of the allocated compute nodes is generally at least 75%. The most impressive speedups are observed on the largest test problems, where compute node efficiencies are consistently around 85%. As with the observed variability in speedups across trials for a single test problem, the pattern of improved speedups with growth in problem size is attributable to

the increased ratio of computation-to-communication time. Although we have considered avenues for further reduction in communication time, specifically in terms of file transfer mechanisms, the resulting efficiency is not raised to above 90%. This barrier is due to the inherent variability in run-time across individual scenario solves at any given PH iteration. Lacking approaches to reduce this variability, which are presently absent in the literature, we believe we are near the upper limits of speedup that can be practically obtained in the case of large test problems. However, the efficiency is already high both in absolute terms and relative to many real-world deployments of parallel algorithms.

In conclusion, the general trends we observe in these experiments indicate that for large problem instances, we can obtain very significant speedups in the execution of our PH algorithm for the SESP on a typical Beowulf compute cluster. Such speedups are crucial for obtaining the required analytic throughput required in the case of the JSF and FCS customers, as these SESP instances are one or more orders of magnitude larger than the largest test case we consider in this paper. In addition to increasing throughput (addressing the issues we raise in Section 4), the resulting speedups allow us to execute PH using parameter settings that tend to yield improved solutions, albeit at the expense of significant increases in serial run-time. For example, the ρ selection rule *sep* generally provides higher-quality solutions than the *f_{100K}* rule; other examples are discussed in Watson et al. (Submitted). Further, large speedups allow us to extend our analysis to SESP’s requiring consideration of much greater numbers of scenarios (e.g., $100 \geq k \geq 200$), e.g., as is required for the FCS logistics system analysis. Finally, we note that our speedups are consistent with those reported by Silva and Abramson (1993) on smaller test problems in a different domain, where the speedups on 18 processors was at most ≈ 14.5 . This equates to roughly 80% efficiency, on a shared-memory machine with significantly lower communication delays than found on a typical Beowulf cluster.

8. Experimental Results: The Amazon EC2 Cluster

Although parallel computing environments are gradually becoming accessible to more and more end-users, the total cost – in terms of expertise, labor, and facilities – of maintaining even a simple cluster is often prohibitive to many analysis departments in both commercial and military organizations. Consequently, a major and often ignored challenge in deploying parallel computing solutions to customers involves the identification and acquisition of a par-

allel computing environment. One alternative to direct procurement is the use of commodity, “pay-as-you-go” computing clusters such as Amazon’s EC2, which we describe below in Section 8.1. One obvious concern with the use of such clusters is whether the raw performance and speedups observed on a native Beowulf cluster will transfer. Experimental consideration of this question is addressed in Section 8.2.

8.1 The Amazon EC2 Compute Cluster

In Section 4 we discussed a variety of benefits that an analyst or decision maker might derive from being able to quickly solve SESP optimizations using parallelized PH. In spite of these sometimes substantial benefits, many organizations do not have the budget to purchase or the skills to manage a large cluster. While our RPC-based implementation of parallel PH would allow a group of workstations to be utilized as a loosely coupled compute cluster, there often are security or organizational considerations that preclude this approach.

In August, 2006, Amazon.com launched the EC2 Elastic Compute Cloud (EC2 Overview, 2007), an on-demand scalable compute grid. From the user perspective, each node in the grid has the compute power equivalent to a 1.7GHz Xeon with 1.75 GB of memory, a 160GB disk, and 250MBs network interconnect. Compute nodes execute virtual machines running under Xen (2007), executing any operating system that can operate on a Xen-hosted virtual machine. EC2 provides a series of *Amazon Machine Images*, or AMIs, that can be booted on EC2 nodes, and further provides a method for generating custom operating system images from scratch. An image executing on a compute node is referred to in EC2 as an *instance*. Charging is based on how many instances are executing, independent of whether they are consuming CPU cycles. Instances are billed at \$0.10 / hour, rounded to the next integral hour, plus \$0.10/GB for data transmitted to EC2 and up to \$0.18/GB for data transferred out. Although EC2 is currently in limited beta release, we experienced no issues in either instantiating runs with 30 instances, or with data transfer and instance reliability.

The EC2 instances have no persistent memory that lasts beyond the life of the instance – shutting down an instance is akin to destroying a machine. In addition to the server farm, Amazon has developed the Simple Storage Service (S3), a massive file system providing data security and redundant backup. Like EC2, it is billed at a very fine grain level. Storage is \$.15/GB month, and data transfer is the same as EC2. Data transfer between EC2 and S3 is free. AMIs must be stored on S3 for booting in EC2. One can use S3 to provide persistent

storage between invocations of an instance on EC2. Using software developed by 3rd parties, S3 can be mounted as a file system on EC2, providing simple persistent storage.

EC2 and S3 are managed through commands issued to Amazon servers via HTTP protocols. Plug-ins to the Mozilla Firefox web browser provide graphical interfaces to EC2 and S3, greatly simplifying management activities such as starting and stopping instances, or moving files. Python, C#, and Java interfaces to the command set are also available.

Due to licensing issues, almost all of the pre-built AMIs are based on Linux. Thus, a user’s perception of ease of use of the EC2 system is tightly related to the user’s familiarity with Linux or related operating systems. While it is not necessary to modify an AMI in order to implement our RPC-based parallelization, we chose to create a custom bundled image that would start the server at boot time. In order to this, we started with an Amazon-produced, pre-built Fedora Core 4 image and added a script that would execute at boot time. The boot script executes a Python script that causes the booting instance to access the S3 file system to fetch the current executable for our server and to start the server, using an input dataset also fetched from S3. Instructions and tools for bundling a new AMI are provided on the Amazon web site.

Finally, to illustrate the appeal of an Amazon-like commodity compute cluster, we observe that the total cost for all of our experimental analyses on the EC2 cluster – including those reported subsequently in Section 8.2 and additional preliminary experimentation – totaled less than \$200.

8.2 Empirical Speedups on the EC2 Cluster

Num. Bases (n)	Num. Scenarios	Serial Run-Time	Parallel Speedups on $ \mathcal{S} $ Compute Nodes
2	10	10.72	3.48 – 4.22
	30	55.98	10.33 – 10.57
5	10	48.40	6.25 – 7.71
	30	362.64	18.39 – 19.52
10	10	344.62	9.07 – 10.85
	30	2335.54	25.23 – 26.71

Table 3: Speedup statistics for parallel PH on the Amazon EC2 compute cluster, using the ρ selection strategy f_{100K} . Serial run-times are reported in minutes.

To assess the performance of our parallel PH implementation on the EC2 cluster, we replicate the experimental methodology previously introduced in Section 7. The sole exception involves omission of results for the ρ selection rule sep on test problems with $|\mathcal{S}| = 30$;

Num. Bases (n)	Num. Scenarios	Serial Run-Time	Parallel Speedups on $ \mathcal{S} $ Compute Nodes
2	10	51.09	4.33 – 5.60
5	10	260.61	10.34 – 12.39
10	10	2189	12.11 – 12.27

Table 4: Speedup statistics for parallel PH on the Amazon EC2 compute cluster, using the ρ selection strategy *sep*. Serial run-times are reported in minutes.

this exception was driven by the excessive run-times required to establish the serial performance baseline in the case of the $n = 5$ and $n = 10$ instances. The results for the ρ selection rules f_{100K} and *sep* are respectively shown in Tables 3 and 4.

Contrasting the results relative to those reported in Section 7 (Tables 1 and 2), we observe qualitatively similar speedups for both the ρ selection strategies f_{100K} and *sep*. In most cases, the speedups obtained on the EC2 cluster are slightly greater than that observed on our Beowulf cluster; the differences in these cases can be attributed to the faster network interconnect (250 MBs versus 100MBs). Consequently, there is no risk inherent in deploying a parallel solution on the EC2 cluster, despite the potential for performance issues relating to the use of virtual machines and multi-user contention for resources.

Serial run-times are, as expected, larger on the EC2 cluster. The EC2 PH runs execute roughly four times longer than the corresponding runs on our native Beowulf cluster. This difference should strictly be attributable to differences in the CPUs; 2.4GHz Pentium 4s in the case of our native Beowulf cluster, and (virtual) 1.7GHz Intel Xeons in the case of the Amazon EC2 cluster. However, we observe a run-time discrepancy roughly twice as large as expected given experiments with physical 1.7GHz Intel Xeon machines. Other users have observed similar mismatches between observed performance and that claimed by Amazon, e.g., see (EC2 Performance Issue, 2007), and we are in communication with Amazon to resolve the issue. In any case, it is clear that absolute throughput relative to that obtained with a native Beowulf cluster will be significantly improved, although to a lesser degree than is possible once the CPUs offered by commodity clusters such as EC2 achieve near-state-of-the-art individual performance levels.

9. Conclusions

The Support Enterprise Sustainability Problem (SESP) is a major, high-impact optimization problem in the analysis and deployment of both commercial and military systems. Previ-

ously, we introduced a novel simulation-based formulation of this problem as a stochastic mixed-integer program (SMIP), and developed an effective serial implementation of Progressive Hedging for generating high-quality solutions. While achieving tractable run-times, e.g., on the order of hours to tens of hours, a number of factors drive the need for parallelization of the Progressive Hedging algorithm in order to achieve significant reductions in run-time; these include the iterative nature of problem and optimization specification, the desire to perform “what-if” analyses, and a significant expected growth in the number of scenarios that much be considered.

We performed a straightforward implementation of Progressive Hedging in parallel, and examined empirical speedups on two compute clusters: a typical Beowulf cluster and the Amazon EC2 cluster. Our objective was to demonstrate that large-scale stochastic programs can in fact be solved on commodity parallel computing hardware, using simple, widely understood technologies for inter-node communication. In practice, end-users typically do not have access to supercomputing-class environments, and acquisition and maintenance of even “simple” clusters is a non-trivial investment. Our experimental results indicate that for large-scale problem instances, near-linear speedups (e.g., efficiencies of roughly 85% or greater) can be obtained using commodity technologies. Obtaining near-linear speedups on the EC2 cluster is particularly significant, as it eliminates the need for end-user acquisition and maintenance of cluster computing technologies, and provides the further benefit of a “pay-as-you-go” cost model.

Fundamentally, there is admittedly little algorithmic novelty in our approach. The “embarrassingly parallel” approach to solving the SESP formulation using Progressive Hedging is effective, obtaining near-linear speedups, so there was little initial motivation for pursuing more advanced approaches; we do plan to examine asynchronous variants of Progressive Hedging in the future (Somervell, 1998), to determine if linear speedups can indeed be obtained. Rather, our focus is on demonstrating the impact of the approach. We are able to solve very large-scale, real-world stochastic programs (the largest of which are not reported here due to the proprietary nature of the data), and have demonstrated that the performance of the approach will transfer to parallel computing environments that are most easily accessible to end-users. The resulting speed-ups significantly improve analytic throughput, with the aforementioned benefits to end-users and improvements in our ability to address more realistic and complicated formulations of the SESP.

Acknowledgments

Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

References

- AMPL. 2007. AMPL. <http://www.ampl.com>.
- Beowulf Overview. 2007. Wikipedia Beowulf entry. http://en.wikipedia.org/wiki/Beowulf_cluster.
- Berland, N.J., K.K. Haugen. 1996. Mixing stochastic dynamic programming and scenario aggregation. *Annals of Operations Research* **64** 1–19.
- Bloomer, J. 1992. *Power Programming with RPC*. O'Reilly and Associates.
- EC2 Overview. 2007. Amazon EC2. <http://www.amazon.com/gp/browse.html?node=201590011>.
- EC2 Performance Issue. 2007. Amazon. <http://developer.amazonwebservices.com/connect/thread.jspa?threadID=16912&tstart=0>.
- Gosavi, A. 2004. *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*. Springer.
- Greenberg, H. 2007. A fine-grained mixed-integer programming model for logistics optimization. Tech. rep., Sandia National Laboratories.
- ILOG. 2007. ILOG CPLEX 10.1. <http://www.ilog.com/cplex>.
- Kall, P., S.W. Wallace. 1994. *Stochastic Programming*. Wiley, Chichester.
- Listes, O., R. Dekker. 2005. A scenario aggregation based approach for determining a robust airline fleet composition. *Transportation Science* **39** 367–382.
- Løkketangen, A., D. L. Woodruff. 1996. Progressive hedging and tabu search applied to mixed integer (0,1) multistage stochastic programming. *Journal of Heuristics* **2** 111–128.

- Maarten, H., M.H. van der Vlerk. 1996–2003. *Stochastic Integer Programming Bibliography*. WWW, <http://mally.eco.rug.nl/biblio/stoprog.html>.
- Muckstadt, J.A. 2005. *Analysis and Algorithms for Service Parts Supply Chains*. Springer.
- Mulvey, J.M., H. Vladimirou. 1991. Applying the progressive hedging algorithm to stochastic generalized networks. *Annals of Operations Research* **31** 399–424.
- Parija, G.R., S. Ahmed, A.J. King. 2004. On bridging the gap between stochastic integer programming and mip solver technologies. *INFORMS Journal on Computing* **16**.
- Rockafellar, R.T., R. J-B. Wets. 1991. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research* 119–147.
- RPC Overview. 1997. Linux Journal. <http://www.linuxjournal.com/article/2204>.
- RPC Specification. 1988. RPC: Remote procedure call protocol specification version 2, rfc1057. <ftp://ds.internic.net/rfc/rfc1057.txt>.
- Sherbrooke, C.C. 1968. METRIC: A multi-echelon technique for recoverable item control. *Operations Research* **16** 121–141.
- Silva, A., D. Abramson. 1993. Computational experience with the parallel progressive hedging algorithm for stochastic linear programs. *Proceedings of the 1993 Parallel Computing and Transputers Conference*. 164–174.
- Slay, F.M. 1984. VARI-METRIC: An approach to modeling multi-echelon resupply when the demand process is poisson with a gamma prior. Tech. Rep. AF501-2, Logistics Management Institute, Washington, D.C.
- Slay, F.M., R.M. King. 1987. Prototype aircraft sustainability model. Tech. Rep. AF601-R2, Logistics Management Institute, Washington, D.C.
- Smith, V.D., D.G. Searles, B.M. Thompson, R.M. Cranwell. 2006. SEM: Enterprise modeling of JSF global sustainment. *Proceedings of the 37th Conference on Winter Simulation*. 1324–1331.
- Somervell, Michael. 1998. Progressive hedging in parallel. *Proceedings of the 33rd ORSNZ Conference*.

Watson, J.P., D.L. Woodruff, D.R. Strip. Submitted. Progressive hedging innovations for a stochastic spare parts support enterprise problem, Sandia National Laboratories Technical Report 2007-3722J.

Xen. 2007. Xensource. www.xensource.com.