Performance Optimizations for a Parallel, Two Stage Stochastic Linear Program
A Case Study of the Military Aircraft Allocation Problem

Akhil Langer
Dept. of Computer Science
University of Illinois at Urbana-Champaign
alanger@illinois.edu

Ramprasad Venkataraman
Dept. of Computer Science
University of Illinois at Urbana-Champaign
ramv@illinois.edu

Udatta Palekar
College of Business
University of Illinois at Urbana-Champaign
palekar@illinois.edu

Laxmikant Kalé
Dept. of Computer Science
University of Illinois at Urbana-Champaign
kale@illinois.edu

Steven Baker
MITRE Corporation
sbaker@mitre.org

ABSTRACT

The US air fleet is tasked with the worldwide movement of cargo and personnel. Due to a unique mixture of operating circumstances, it faces a large scale and dynamic set of cargo movement demands with sudden changes almost being the norm. Airfleet management involves periodically allocating aircraft to its myriad operations, while judiciously accounting for this uncertainty to minimize operating costs. We have formulated this allocation problem as the optimization of a stochastic two-stage integer program.

Our work aims to enable rapid decisions via a scalable parallel implementation. We present the design of our parallel solution that applied a well-known master- worker approach to solving the undecomposed linear programs underlying the formulation. This paper presents a narrative of the issues encountered and solutions that we developed to combat these. We believe these techniques may be generally applicable in other contexts where a parallel solution of stochastic optimization is of interest.

Categories and Subject Descriptors

G.1.6 [**Mathematics of Computing**]: Numerical Analysis—*Optimization: Stochastic Programming*; I.6 [**Simulation and Modeling**]: Applications—*airfleet management*; D.1.3 [**Software**]: Programming Techniques—*Parallel Programming*; J.1 [**Computer Applications**]: Administrative Data Processing—*Military*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WOODSTOCK '97 El Paso, Texas USA
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$10.00.

General Terms

Algorithms, Design, Performance

Keywords

Stochastic optimization, High Performance Computing, Simulation, Airfleet Management

1. INTRODUCTION

Stochastic optimization plays an important role in the analysis, design, and operation of modern systems. It provides a means of coping with inherent system noise and models that are highly nonlinear, high dimensional, or otherwise inappropriate for classical deterministic methods of optimization. In the classical deterministic models, the solutions obtained are optimal for the specific problem but may not be optimal for the situation that actually occurs. Being able to take this randomness into account is critical for many problems where the essence of the problem is dealing with the randomness in some optimal way. Stochastic programming enables the modeller to create a solution that is optimal over a set of scenarios.

Stochastic optimization algorithms have broad application to problems in statistics, science, engineering, and business. Specific applications include business (making short- and long-term investment decisions in order to increase profit), transportation (planning and scheduling logistics), aerospace engineering (running computer simulations to refine the design of a missile or aircraft), medicine (designing laboratory experiments to extract the maximum information about the efficacy of a new drug), and traffic engineering (setting the timing for the signals in a traffic network). There are, of course, many other applications in agriculture, energy, military, telecommunications, water management etc. In this paper, we propose and evaluate novel techniques to solve a 2-stage stochastic optimization model with linear recourse for an aircraft planning problem.

At a high level, stochastic optimization involves searching for an solution that is optimal over a set of scenarios.

In most cases, it would involve evaluation of the candidate solution for each scenario. Since, the number of scenarios can be large, it behooves us to consider strategies to reduce this work. We analyze different strategies for reducing the stage 2 work, namely advanced start and scenario clustering and show that advanced start with random scenario clustering reduces the number of iterations and the time to convergence. We also present an LRFU based scheme for cut management in stage 1, that performs better than cut management based on just the frequency or recency of their activity. We then present our scalability results using these optimization strategies.

Stochastic optimization algorithms have been growing rapidly in popularity over the last decade or two, with a number of methods now becoming "industry standard" approaches for solving challenging optimization problems. Given the complexity of the systems and the scale at which the system parameters change, the time to solve stochastic models is very critical. It is therefore natural to develop techniques to utilize the parallel computing resources (in the form of multi-core desktops and servers, clusters, super-computers) to cut down the solution times. An interesting development in this context, is the widespread availability of cloud computing platforms which offer computational resources and optimization solvers as services. Since users pay for the time they use these resources/services, it is critical to optimize the application. We strongly believe that this area has great potential for research in parallel computing community and hope that it would receive due attention by the community.

Contents of the paper are organized as follows. In Section 2 we briefly introduce the problem of concern to the authors - the United States military aircraft allocation problem managed by TACC. In Section 3 we discuss our parallel program design for the benders decomposition approach. In Section 4,5 we present our optimization strategies for the stage 1 problem while in Section 6 we present our analysis of the stage 2 problem. Scalability results are presented in Section 7. Related work is reviewd in Section 8. Finally, conclusion and future work are given in Section 9.

2. MODEL FORMULATION AND THE SOLUTION APPROACH

In this section, we give an overview of the United States military aircraft allocation problem. We then present our stochastic linear program formulation of the problem. United States Air Mobility Command (AMC) [1] manages a fleet of over 1300 aircrafts [1] that operate globally. Various missions of AMC involve uncertainty based on worldwide tension and commitments. Aircrafts are allocated by the military at its different bases in anticipation of the demands for several missions to be conducted in the period of one month or fifteen days. Aircraft breakdowns, weather, natural disaster, conflict, are some of the various possible outcomes that confound decision support. The purpose of the stochastic model is to optimally allocate aircraft against each mission area in a manner that minimizes subsequent disruption. The planning period occurs prior to the execution period, where stochastic cargo and mission realizations drive actual mission requirements.

[1]http://www.amc.af.mil/

TACC[2] is responsible for allocating aircrafts to three of the primary mission types flown by AMC:

- Channel missions, which are regularly scheduled missions between the US and overseas locations
- Contingency missions, which are irregularly scheduled missions that deliver cargo to an international "hot spot," and
- Special assignment airlift missions (SAAMs), which are chartered by military units for a specific purpose.

Aircrafts are allocated by aircraft type, airlift wing, mission type and day. In situations when self-owned military aircrafts are not sufficient for outstanding missions, civilian aircrafts are leased. Rent of civilian aircrafts procured in advance for the entire planning cycle is lower than the rent of civilian aircrafts leased at short notice. Therefore, a good prediction of the aircraft demand prior to the schedule execution reduces the execution cost. We model the allocation process as a two-stage stochastic linear program with complete recourse.

In the *first stage*, before a realization of the demands are known, one has to make a decision about leasing of the long term civilian aircrafts and the allocation of the aircrafts to different missions at each base location.

$$\min \quad Cx + \sum_{k=1}^{K} p_k \theta_k \tag{1}$$

$$s.t. \qquad Ax \leq b, \tag{2}$$

$$E_l x + \theta \leq e_l \tag{3}$$

In the objective function(1), x corresponds to the aircraft allocations by the aircraft type, location, mission and time. C is the cost of allocating military aircrafts and leasing of civilian aircrafts. θ_k and p_k are the cost and probability of occurrence of scenario k, respectively. Constraints in (2) are the feasibility constraints, while constraints in (3) are the optimiality constraints obtained from stage 2 and are also called as cuts.

At the *second stage*, the expected cost of an allocation is computed by solving the second stage optimization problem for a large number of realizations(scenarios).

$$\min \qquad {q_k}^T y \tag{4}$$

$$s.t. \quad Wy \leq h_k - T_k x \tag{5}$$

Second stage optimization helps the stage 1 to take the recourse action of increasing the capacity for satisfying an unmet demand by providing cuts(6).

$$\theta_k \leq \pi_k * (h_k - T_k x^*) - \pi_k T_k (x - x^*) \tag{6}$$

where π_k are the dual multipliers obtained from stage 2 optimization and x^* is the allocation vector obtained from the last stage 1 optimization.

Note that our formulation of the aircraft allocation model has complete recourse (i.e. all allocations generated in stage 1 are feasible) because any demand that cannot be satisfied by allocation done in stage 1 is met by short term leasing of civilian aircrafts at a high cost. Because of space limitation, detailed description of the model is not possible here. For complete description of the model and the cost comparison of stochastic vs deterministic models, please refer to [15].

[2]Tanker Airlift Control Center, http://www.618tacc.amc.af.mil

Table 1: Airlift Fleet Assignment Models Set

Model Name	#Stg1 variables	#Stg2 Linear Variables Per Scenario	#Stg2 Constraints Per Scenario
3t	135 + #scenarios	8970	5572
5t	225 + #scenarios	13862	8869
10t	450 + #scenarios	25573	16572
15t	675 + #scenarios	34642	23375
30t	1350 + #scenarios	66304	46445

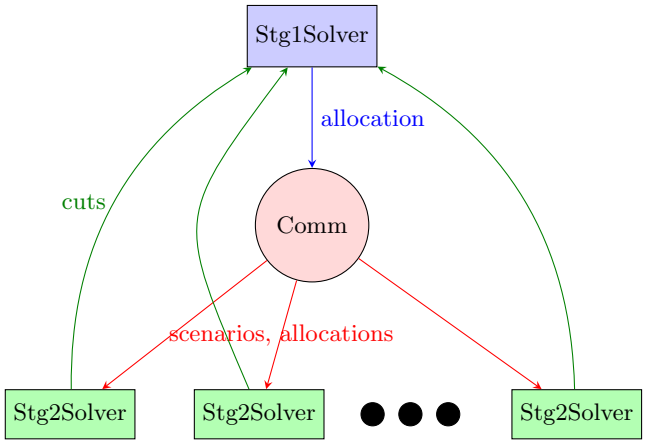


Figure 1: Design

3. PARALLEL PROGRAM DESIGN

We implemented the solver for the two-stage bender's decomposition formulation in Charm++ [10–13]. Charm++ is an object-oriented parallel programming framework based on C++. It has an adaptive run-time system. *Chares* and *chare arrays* form the basic unit of program control flow. *Chares* are allocated to processors and can be migrated from one processor to another. Computation is done by either remotely or locally invoking *entry methods* on a *chare*.

Figure 1 gives a schematic of the implementation of our solver. It has three entities, namely the stage 1 solver, the communicator (a.k.a. *Comm*) and the stage 2 solvers. Stage 1 solver (the master) proposes new aircraft allocations based on the feedback it has collected from stage 2 so far and sends it to the *Comm*. *Comm* assigns scenarios to the stage 2 solvers. Stage 2 solvers evaluate the scenarios assigned to them and send the feedback(cuts) to the stage 1 solver. The *Comm* and the stage 1 solver are chares and are assigned dedicated processors. For each of the remaining processors, there is one chare array element acting in the role of a stage 2 solver.

4. ADVANCED STARTS IN STAGE 1

We first evaluate the effect of advanced start(or warm start) in stage 1 optimization. With advanced start, the LP solver starts with the solution and basis from the previous solve, which in this case is the solve from the last iteration. Since, in every iteration the LP is changed only by the addition of some constraints, the optimization of the new LP takes much less iterations as comapred to the solves done from scratch. Figure 2 validates the advantage of advanced start for this problem.

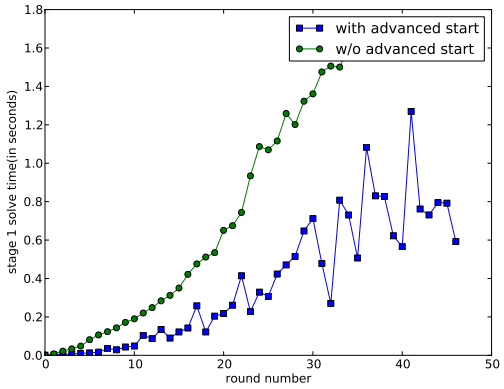


Figure 2: Stage 1 solve times with and without advanced start

However, an important thing to note here is that the stage 1 solve time increases with the iteration number irrespective of the use of advanced start. Additionally, the memory requirements of the stage 1 model increases as the number of rounds increase because of the accumulation of cuts in every round. For large stage 1 problems, which take many iterations to converge, the increasing stage 1 solve time and the increasing memory demands poses a serious serial bottleneck making these problems intractable with the Bender's approach. Figure 3 shows this effect observed in a relatively larger 15t model solved upto 1% accuracy. The memory usage is as high as 5GB and the stage 1 solve time can go upto 100secs. In the following section on stage 1 optimization(Section 5), we address these bottlenecks by employing intelligent cut management schemes.

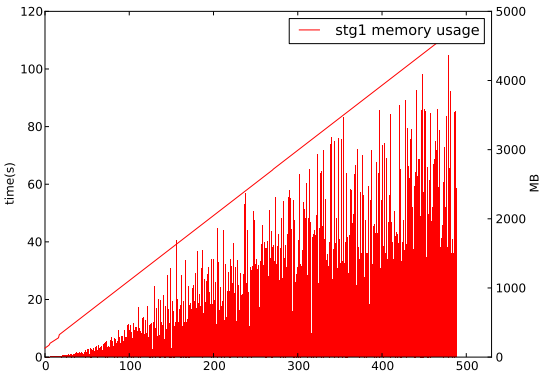


Figure 3: Stage 1 memory usage and solve time for 15t model

5. RETAINING THE MOST USEFUL CUTS IN STAGE 1

Certain cuts become inactive in the course of convergence either because they present weak optimiality constraints and have been superseded by stronger optimiality cuts or because they were feasibility constraints but now there are other cuts that encompass the feasibility posed by them. Such cuts simply add to the size of the stage 1 model and its solve time and can be safely discarded. Activity of a cut in an optimization is determined by the slack value of it's constraint. If the slack value is zero (within the tolerance limits) then the cut was active (or useful), otherwise it was not used in finding the optimal value to the linear program. Figure 4 shows the usage rate (defined by equation 7) of the cuts generated during the course of convergence of a 5t model. Most of the cuts have very low usage rate while a significant number of the cuts were not used at all.

$$\text{Cut Usage Rate} = \frac{\text{number of rounds in which it is used}}{\text{total number of rounds since its generation}} \quad (7)$$

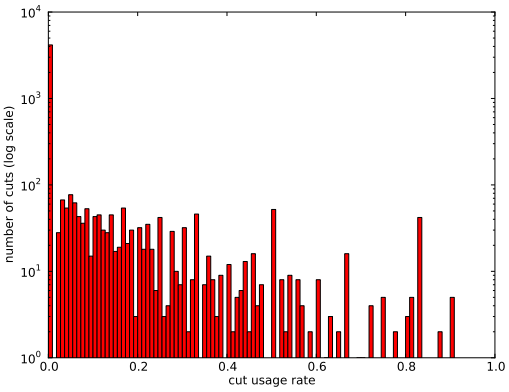


Figure 4: Cut usage rate for a $5t$ model

We therefore implemented a cut retirement scheme, that discrads/retires cuts when the total number of cuts in the stage 1 model exceed a predefined limit set on the maximum number of cuts that can be stored in stage 1 model. Cuts with smaller usage rate (defined by Equation 7) are discarded. The rationale behind safely discarding cuts is that if any cut that might be needed in future rounds gets discarded it can be regenerated again by the feedback providing mechanism implicit in the bender's approach. This can cause an increase in the number of rounds required to reach convergence, but this approach guarantees bounded stage 1 solve time and memory requirements of the stage 1 model. Figure 5 demonstrates all these effects very clearly. After every round of the bender's method, cut score is updated based on it's activity in that round. The extra X low scoring cuts can be determined using a partial sort that runs in linear time.

An input parameter to the cut retirement scheme is the *Cut Window*. *Cut Window* is the upper limit on the number of cuts allowed in stage 1 model. For ease of readability, it is defined as the maximum number of cuts divided by the number of scenarios. Therefore, *Cut Window* of 10 in a model with 120 scenario means a maximum of $120 * 10$ or 1200 are stored in the stage 1. Runs with different *Cut Windows* will

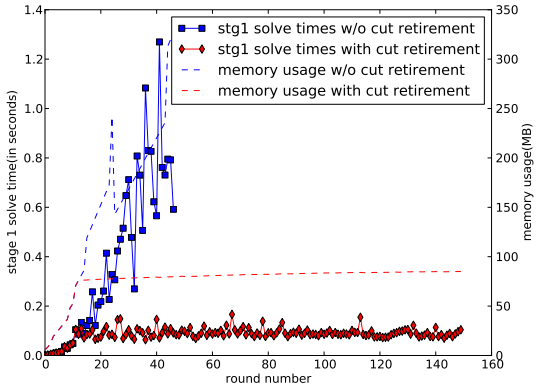


Figure 5: Comparing the stage 1 solve times and memory usage with and without cut retirement

take different number of rounds to converge. In Figure 6 and 7, we study the effect of different *Cut Windows* on the number of rounds required to converge and also the time to solution. As we decrease the *Cut Window* size, it takes increasing number of rounds to converge but the time to solution decreases. Smaller *Cut Windows* reduce the individual stage 1 solve time, leading to an overall improvement in the time to solution even though it takes larger number of rounds to converge. Decreasing the *Cut Window* beyond a certain limit, leads to significant increase in the number of rounds because of many potential cuts getting discarded earlier and need to be regenerated in later rounds. Even further decrease in the *Cut Window* makes the problem impossible to converge because of insufficient cut collection at any time. Hence, these experiments demonstrate the need to make a wise choice on the *Cut Window* size to get fastest time to solution. e.g. for 5t model with 120 scenarios, optimal *Cut Window* size is close to 25 while for a 10t model with 120 scenarios it is close to 15.

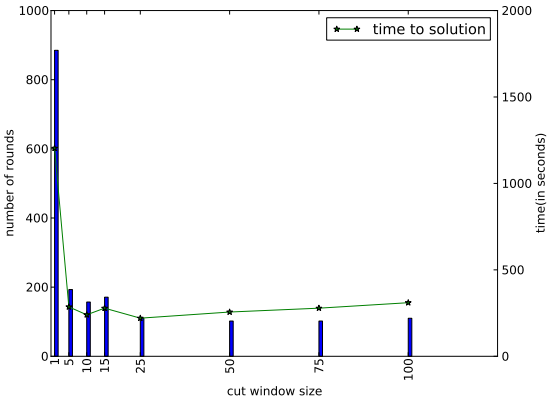


Figure 6: Performance with different *c*ut-windows for $5t$ model (solved upto 0.1% convergence on 8 cores)

We investigate cut management further to study it's performance with other cut scoring schemes. Three cut scoring

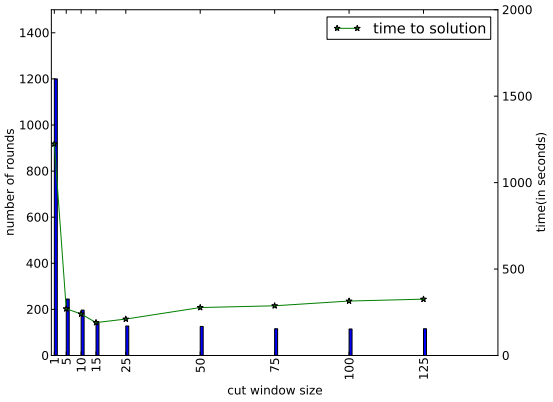


Figure 7: Performance with different *cut-windows* for $10t$ model (solved upt 1% convergence on 32 cores)

schemes are discussed here namely, the least frequently used, the least recently used and the least recently/frequently used. Each one of these are briefly discussed here:

- *Least Frequently Used (LFU)* - A Cut is scored based on it's rate of activity since it's generation.

$$LFU\_Score = \frac{\text{Number of rounds in which it was active}}{\text{Number of rounds since its generation}}$$

  This scoring method was used for results presented in Figure 6 and 7.

- *Least Recently Used (LRU)* - In this scheme, recently used cuts are scored higher. Therefore, score of a cut is simply the last round in which it was active.

$$LRU\_Score = \text{Last active round for the cut}$$

- *Least Recently/Frequently Used (LRFU)* - This scheme takes both the recency and frequency of cut activity into account. Each round in which the cut was active contributes to the cut score and the contribution is determined by a weighing function $\mathcal{F}(x)$, where x is the time span from the activity in the past to current time.

$$LRFU\_Score = \sum_{i=1}^{k} \mathcal{F}(t_{base} - t_i)$$

  where $t_1, t_2, ..., t_k$ are the active rounds of the cut and $t_1 < t_2 < ... < t_k \leq t_{base}$. This policy can demand large amount of memory if each reference to every cut has to be maintained and also demands a considerable amount of computation every time the cut retirement decisions are to be made. Lee, et. al. [14,16] have proposed a weighing function $\mathcal{F}(x) = (\frac{1}{p})^{\lambda x}$ $(p \geq 2)$ which reduces the storage and computational needs drastically. They tested it for the cache replacement policy and obtained competitive results. With this weighing function, the cut score can be calculated as follows:

$$S_{t_k} = \mathcal{F}(0) + \mathcal{F}(\delta) S_{t_{k-1}},$$

  where S_{t_k} is the cut score at the kth reference to the cut, $S_{t_{k-1}}$ was the cut score at the $(k-1)$th reference and $\delta = t_k - t_{k-1}$. For more details and proofs for the weighing function refer to [14]. We use $p = 2$ and $\lambda = 0.5$ for our experiments.

Figure 8 compares the result of these strategies. LRFU gives the best performance out of the three strategies. *Cut Window* used for these experiments was the optimal values obtained from experiments in Figure 6 and 7.

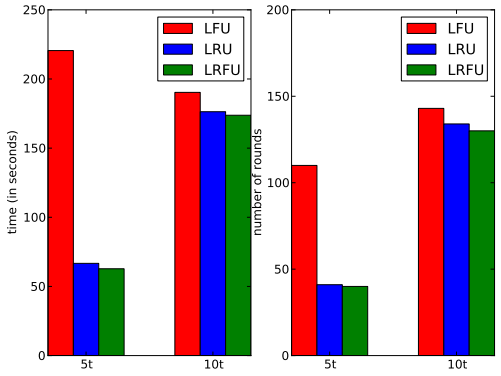


Figure 8: Performance of different cut scoring strategies for $5t$(8cores, cut-window=25, 0.1% convergence) and $10t$ (32 cores, cut-window=15, 1% convergence)

Finally to conclude this section, Figure 9 shows the benefit of cut management on the stage 1 memory usage and solve times of the 15t model solved upto 1% accuracy. The total time to solution reduces from 19025s (Figure 3) without cut retirement to 8184s (Figure 9) with cut retirement, about 57% improvement.

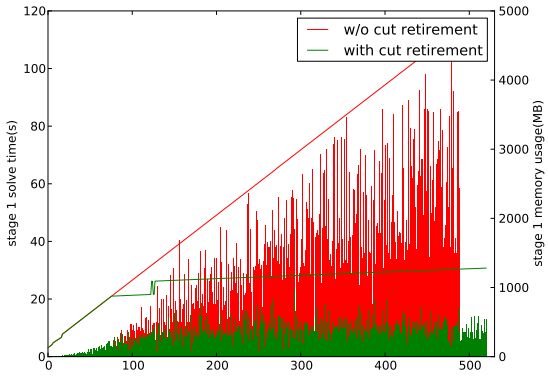


Figure 9: Stage1 solve time and memory usage for $15t$ model solved upto 1% convergence with *cut-window* of 75

6. EXPLOITING ADVANCED STARTS IN STAGE 2

- talk about how using advanced start in stage 2 can also reduce the stage 2 solve time and discuss different ways to reduce the stage 2 solve time: by custering scenarios using demands, duals, etc.
- Since, scenario differs only in demands, it is easy to cluster them

In every iteration, there are as many stage 2 LP solves as there are scenarios. This constitutes a major portion of the computation involved in the Bender's approach because of the large number of scenarios in practical applications. Even small amount of reduction in the number of rounds or average stage 2 solve time can have sizeable payoffs. In this section, In this section, we analyze different strategies to reduce the amount of time spent in doing stage 2 work.

The demands for the concerned military aircraft allocation proble have probabilisitc distribution. Scenarios were generated with channel demands drawn from a *normal* distribution, SAAM demands from a *poisson* distribution and the contingencies occurring randomly with some assigned probabilities of occurrence.

At first, we employed a pull-based mechanism to assign work to stage 2 solvers i.e. as soon as a stage 2 solver becomes idle, it sends a work request to the stage 2 manager which assigns it an unevaluated scenario. In contrast to the stage 1 lp solves, stage 2 lp solves take more time with the advanced start feature as compared to fresh start. This can happen because the initial basis from the previous scenario solve can be a bad starting point for the new scenario. Despite the slower lp solves with the advanced start our experiments show that runs with advanced start take fewer rounds to converge than with fresh start (term *frest start* is synonymous with *w/o advanced start*). This indicates that starting from the previous solves gives us better cuts. This behavior was seen in other models as well. Reason for such a behavior is yet to be studied. Figure 10 shows that runs with stage 2 reset (in red color) took 300 rounds to converge as compared to just $100 - 150$ rounds with advanced start. Consequently, the time to solution with advanced start is much less than with stage 2 reset despite slower stage 2 lp solves. Therefore, from hereon we will continue to use the advanced start feature. An important point to note from Figure 10 is that advanced start together with the pull-based work assignment mechanism introduces variability across various runs of the same program. Figure 10 shows the number of rounds and time to solution for 25 runs on the same model. Scenario assignment to solvers takes place based on the order in which the work requests are received, which can vary across different runs because of variable message latencies and also fine variations in LP solve times because of system noise. A different order of LP solves on a solver yields different cuts for identical scenario evaluation because the solver starts from the basis of the previous scenario evaluation. This variation in cuts affects the course of convergence of the model. Variation across different runs can be significant making it difficult to measure the effect of different optimization strategies. Additionally in many situations, getting an optimal solution in predictable time is considered more important than getting it in the shortest possible time. Therefore, removing the variability is an important consideration. Turning off the advanced start feature can signficantly increase the time to solution and hence is not a good idea, however the scenario evaluation order can be pre-determined by assigning a fixed set of scenarios to each solver. This approach can potentially decrease the processor utilization because of load imbalance as it does not account for the LP solve times and just assigns equal number of scenarios to each solver. However, traces show that the processor utilization with pull-based and push-based work assignment schemes distrurbs the load balance only minutely. Hence, push based scheme is a win-win approach as it eliminates the undesirable variability in the solution times.

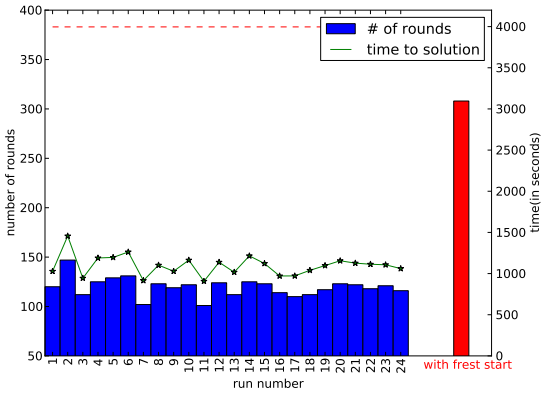


Figure 10: Variation across runs with advanced-start and their comparison with fresh start ($10t$ model)

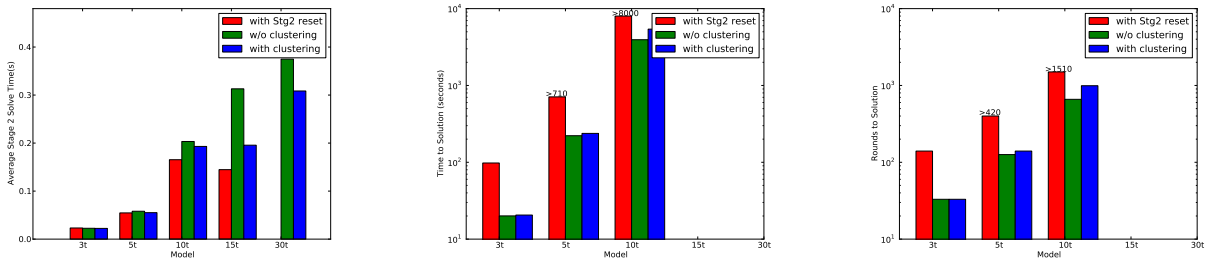
Since, the scenarios are related to each other, it is possible to use the optimal solution of one scenario to compute the optimal solution for the next scenario. We now suggest a clustering based scenario assignment scheme. The advanced start feature can lead to reduction in solve times when similar scenarios are solved consecutively. We can use this property to speedup stage 2 solves. Similarity between scenarios can be determined either by using the demands in each scenario or the duals (cuts) returned by them or by a hybrid of demands and duals. In this paper, we use the demands to cluster scenarios because they are known apriori i.e. before the stochastic optimization process begins. We use the well known KMeans [9] algorithm for clustering scenarios. Since, the clusters returned from KMeans can be unequal in size, we use a simple approach (described in Algorithm 6) to migrate some scenarios from oversized clusters to the undersized clusters.

Figure 6 compares the improvement in average stage 2 solve times when scenarios are clustered using Algorithm 6.

However, our results show that random clustering gives faster convergence rate than KMeans clustering(Figure 12). Therefore, despite faster stage 2 solves, clustering does not gives us faster time to solution(Figure 13). These experiments lead to the conclusion that advanced start with random clustering is the best combination for faster time to solution.

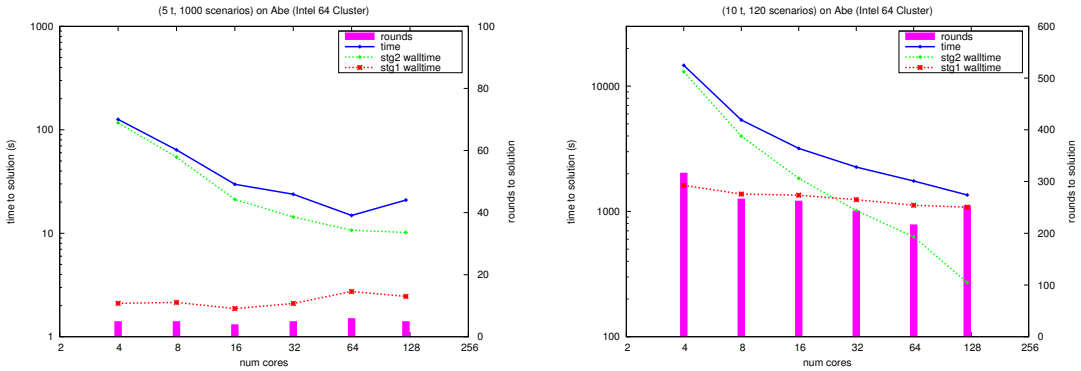
7. SCALABILITY

With the optimizations stated above we were able to achieve significant speed-ups upto 122 cores. For 120 scenarios, a



(a) (b) (c)

Figure 13: Comparing the effect of advanced start and clustering on performance of different models



(a) (b)

Figure 14: Scalability plots for $5t$ and $10t$ model solved upto 0.1% convergence

Algorithm 1 The Scenario Clustering Algorithm

Input
D_i- Demand set for scenario i $(i = 1, 2, ..., n)$
k - number of clusters

Output
Algorithm
{label, centroids} = kMeans($\{D_1, D_2, D_3, ..., D_n\}$, k)
IdealClusterSize = $\frac{n}{k}$
$size_i$ = size of cluster i
{Identify Oversized clusters}
$\mathcal{O} = \{c \in Clusters \mid size_c > IdealClusterSize\}$
{Identify Undersized clusters}
$\mathcal{U} = \{c \in Clusters \mid size_c < IdealClusterSize\}$
$\mathcal{S}$: set of adjustable points
for $c \in \mathcal{O}$ **do**
  Find $(size_i - IdealClusterSize)$ points in cluster c that are farthest from $centroid_c$ and add them to the set $\mathcal{S}$
end for
while $size(\mathcal{S}) > 0$ **do**
  Find the closest pair of cluster $c \in (U)$ and point $p \in \mathcal{S}$

  Add p to cluster c
  Remove p from $\mathcal{S}$
  if $size_c == IdealClusterSize$ **then**
    Remove c from $\mathcal{U}$
  end if
end while

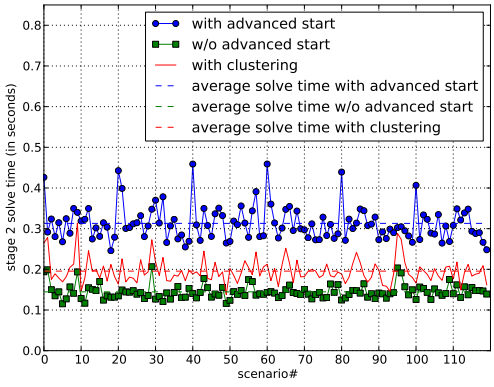


Figure 11: Comparison of average stage 2 solve time between stage 2 reset, random clustering and clustering using KMeans algorithm

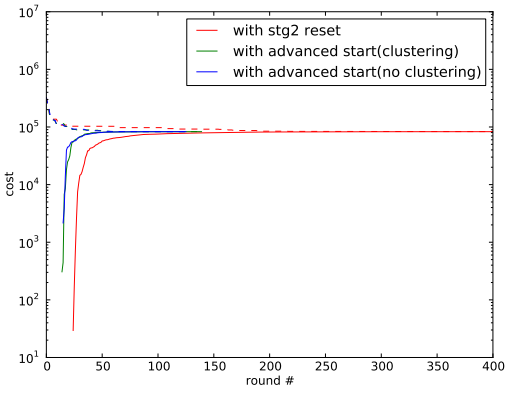


Figure 12: convergence rate comparison b/w stg2 reset, clustering and no clustering

maximum of only 122 processors are required - 1 stage 1 solver, 1 *Comm* and 120 stage 2 solvers for 5t and 10t model respectively. Figure 14(a) and 14(b) show the scalability plots with stage 1 and stage 2 walltime breakdown. The plots also demonstrate the Amdahl's effect as the maximum parallelism available is proportional to #scenarios that can be solved in parallel.

8. RELATED WORK

Stochastic linear programs can be solved using the extensive formulation(EF) [7]. Extensive formulation of a stochastic program is its deterministic equivalent program in which constraints from all the scenarios are put together in a single large scale linear program. e.g. the extensive formulation for a stochastic program corresponding to stage 1 given in equations 1, 2, 3 and stage 2 in equations 4, 5 can be written as:

$$\begin{aligned} \min \quad & c^T x + \textstyle\sum_{k=1}^{K} p_k q_k^T y_k \\ s.t. \quad & Ax = b, \\ & T_k x + W_{y_k} = h_k, \qquad k = 1, ..., K \\ & x \geq 0, y_k \geq 0, \qquad k = 1...., K \end{aligned}$$

EF results in a large linear program which can be solved using any liner program solver but because LP solvers are hard to parallelize, other parallelization approaches are sought for. Recently, there has been some work on parallelization of the simplex algorithms for linear programs with dual block-angular structure [5]. Lubin et.al.in [20], demonstrated how emerging high-performance computing architectures can be used to solve certain classes of power grid problems, namely, energy dispatch problems. Their PIPS solver is based on the interior-point method and uses a Schur complement technique to obtain a scenario-based decomposision of the linear algebra.

J. Linderoth, et. al. [17,18] have studied the performance of two-stage stochastic linear optimizations using the L-shaped algorithm on computational grids. Unlike modern supercomputers, computational grids have poor communication latencies and availability of nodes is sporadic. Hence, their work focusses on performance of an asynchronous approach to the bender's decomposition. In contrast, our work is based on a synchronous approach where a new iteration is initiated only after completion of all the scenario solves from the previous iteration. They have also applied regularization by means of trust region(see [18]) to improve the convergence rate.

9. CONCLUSION AND FUTURE WORK

For stochastic optimization with Bender's approach, stage 1 presents a serial bottleneck that seriously inhibts the efficieny of any parallel implementation. We presented a *LRFU* based cut management scheme, that completely eliminates the memory bottleneck and significantly reduces the stage 1 solve time, thus making the optimizaiton of large scale problems such as 15t and 30t tractable. Most of the stochastic programs incorporate large number of scenarios to hedge against all possible uncertainties. Therefore, stage 2 work constitutes a significant portion of the total work done in stochastic optimizations. We analyzed different dimensions to stage 2 optimzation and concluded that advanced start with random scenario solve ordering can speed up the rate of convergence of the stochastic program. Even though scenario clustering reduces the average stage 2 solve time but it takes more rounds to converge. With these optimizations, we were able to obtain a speedup of upto X% for the 10t problem with 120 scenarios. We believe that these optimization strategies can be applied to other sotchastic programs in general.

A lot of future work ensues from this research. The proposed strategies can be applied to other stochastic linear problems e.g. industrial design, environmental planning, asset management, etc. A list of test problems for stochastic linear programming is available at [6]. All of our experiments were with the use of Gurobi [3] LP solver, we would also like to test these optimizations with other popular linear program solvers such as CPLEX [4], GLPK [21], COIN-CLP [2, 19]. These strategies can also be applied to stochastic integer programs that have integer variables in stage 1. Stochastic integer programs have many real-life applications and consitute a significant portion of the broader class of stochastic programs. Motivate Parallel Branch and Bound using these optimizations

10. TERAGRID

The Teragrid/NCSA Intel 64 Cluster *Abe* (installed at NCSA, Illinois) was instrumental in conducting this research. The Gurobi linear program solver is a licensed solver available for free academic use. It requires licensing on a per node basis, making most of the supercomputers unviable for research because of their large node count. With the help of Abe cluster administrators we were able to get a time shared (and dedicated at crucial times!) reservation on a select set of high memory compute nodes of Abe. This made our scalability runs feasible as we had to acquire licenses only for the assigned set of nodes. Runs on *A*be cluster were done under the TeraGrid [8] allocation grant ASC050040N supported by NSF.

11. ACKNOWLEDGMENTS

The research was supported by the MITRE Research Agreement Number 81990 with UIUC.

The author gratefully acknowledge the use of the parallel computing resource provided by the Computational Science

and Engineering Program at the University of Illinois. The CSE computing resource, provided as part of the Taub cluster, is devoted to high performance computing in engineering and science.

12. REFERENCES

[1] Air Mobility Command. Air Mobility Command Almanac 2009. Retrieved 12 Sep 2011. *http://www.amc.af.mil/shared/media/document/AFD-090609-052.pdf*.

[2] CoinOR. COmputational INfrastructure for Operations Research. Software, 2012. `http://www.coin-or.org`.

[3] Gurobi Optimization Inc. Gurobi Optimizer. Software, 2012. `URLhttp://www.gurobi.com/welcome.html`.

[4] IBM. IBM ILOG CPLEX Optimization Studio. Software, 2012. . `http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/`.

[5] Parallel distributed-memory simplex for large-scale stochastic LP problems. Preprint ANL/MCS-P2075-0412, Argonne National Laboratory, Argonne, IL. April 2012. `www.optimization-online.org/DB_HTML/2012/04/3438.html`.

[6] Test Problem Collection for Stochastic Linear Programming. *http://www4.uwsp.edu/math/afelt/slptestset/download.html*, Retrieved April 20th, 2012.

[7] J. Birge and F. Louveaux. *Introduction to stochastic programming*. Springer, 2011.

[8] C. Catlett et al. TeraGrid: Analysis of Organization, System Architecture, and Middleware Enabling New Types of Applications. In L. Grandinetti, editor, *HPC and Grids in Action*, volume 16, pages 225–249, Amsterdam, 2007. IOS Press.

[9] J. Hartigan and M. Wong. Algorithm AS 136: A K-means clustering algorithm. *Applied Statistics*, pages 100–108, 1979.

[10] L. V. Kale, E. Bohm, C. L. Mendes, T. Wilmarth, and G. Zheng. Programming Petascale Applications with Charm++ and AMPI. In D. Bader, editor, *Petascale Computing: Algorithms and Applications*, pages 421–441. Chapman & Hall / CRC Press, 2008.

[11] L. V. Kale and S. Krishnan. Charm++: Parallel Programming with Message-Driven Objects. In G. V. Wilson and P. Lu, editors, *Parallel Programming using C++*, pages 175–213. MIT Press, 1996.

[12] L. V. Kalé, B. Ramkumar, A. B. Sinha, and A. Gursoy. The CHARM Parallel Programming Language and System: Part I – Description of Language Features. *Parallel Programming Laboratory Technical Report #95-02*, 1994.

[13] L. V. Kalé, B. Ramkumar, A. B. Sinha, and V. A. Saletore. The CHARM Parallel Programming Language and System: Part II – The Runtime system. *Parallel Programming Laboratory Technical Report #95-03*, 1994.

[14] C. Kim. Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transactions on Computers*, 50(12), 2001.

[15] A. Langer. Enabling massive parallelism for two-stage stochastic integer optimizations: A branch and bound based approach. Master's thesis, Dept. of Computer Science, University of Illinois, 2011. http://charm.cs.uiuc.edu/media/11-57.

[16] D. Lee, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho, and C. Kim. Lrfu (least recently/frequently used) replacement policy: A spectrum of block replacement policies. In *In IEEE Transactions on Computers*. Citeseer, 1996.

[17] J. Linderoth and S. Wright. Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24(2):207–250, 2003.

[18] J. Linderoth and S. Wright. Computational grids for stochastic programming. *Applications of stochastic programming*, 5:61–77, 2005.

[19] R. Lougee-Heimer. The common optimization interface for operations research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, 47(1):57–66, 2003.

[20] M. Lubin, C. G. Petra, M. Anitescu, and V. Zavala. Scalable stochastic optimization of complex energy systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 64:1–64:64, New York, NY, USA, 2011. ACM.

[21] A. Makhorin. The GNU Linear Programming Kit (GLPK). GNU Software Foundation, 2000. `http://www.gnu.org/software/glpk/glpk.html`.