ENABLING MASSIVE PARALLELISM FOR TWO-STAGE STOCHASTIC INTEGER OPTIMIZATIONS
A BRANCH AND BOUND BASED APPROACH

BY

AKHIL LANGER

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Adviser:

Professor Laxmikant V. Kalé

Abstract

DoD's transportation activities incur USD 11+Billion expenditure anually. Optimal resource allocation under uncertainty provides huge opportunity for improvement and commensurate cost savings. Stochastic optimization techniques are used to incorporate uncertainty in the data to arrive at robust resource allocations. The application of stochastic optimization extends to a broad range of areas ranging from finance to production to economics to energy systems planning. We study the DoD Air Mobility Command's airfleet assignment problem that schedules 1300+ aircrafts to deal with combat delivery, strategic airlift, air refueling, aeromedical evacuation operations, etc. around the world. Our formulation of the airfleet allocation problem for a small time horizon shows that annual average cost benefits of 3% can be obtained by using stochastic integer optimization. Despite the potential cost benefits, use of stochastic integer optimization has remained intractable because the computational complexity of the problem prevents rapid decisions. Modern supercomputers can attain performance of several petaflops and hence can enhance solution tractability for use in a highly dynamic decision environment. We start with a conventional parallel decomposition of the problem, analyze the challenges and address a number of performance optimizations like cut retirement and scenario clustering. We then present results from a novel branch-and-bound based design that converges to an optimal integer allocation for large problems while allowing evaluation of tens of thousands of possible scenarios. We believe that this is an interesting and uncommon approach to harnessing tera/petascale compute power for such problems without decomposing the linear programs further.

Acknowledgements

I would like to thank my advisor Prof. Laxmikant V. Kalé for having confidence in my ability to do this project (given its interdisciplinary nature). His insightful thoughts and ideas, his enthusiasm and energy kept the motivation high throughout the duration of this work.

Prof. Udatta Pralekar (Department of Business Administration, UIUC) and Ram (Parallel Programming Lab, UIUC) have played an important role in the development of the work. In many instances, Ram has been an ear to my doubts and his experience has been useful in making informed decisions that gave direction to the project. I would like to thank Prof. Udatta Palekar for his time and patience while explaining the concepts of Stochastic Optimization. His knowledge of the field kept us confident about the methodology being used to approach the problem. Thanks to Steven Baker (MITRE Corp.) for his help with the stochastic model formulation of the DoD aircraft allocation problem.

My thanks to the colleagues at Parallel Programming Lab (PPL). Their generously extending help in numerous instances, whether related to Charm++ or any other technical issues, has helped in the smooth progress of the work. Most of the results presented in the thesis were done on Taub and Abe clusters maintained by NCSA. Help provided by the cluster administrators was critical to do these runs.

Finally, I would like to thank my teachers and friends who continue to impart principles in me and have been a very strong source of inspiration for me.

Table of Contents

List of Figures

List of Abbreviations

LP	Linear Program
IP	Integer Program
BnB	Branch-and-Bound
stg1	stage 1
stg2	stage 2
Comm	Communicator
searchVtx	Search vertex
S1LB	Stage 1 Load Balancer
S2M	Stage 2 Manager
PE	Physical Processor
SLP	Stochastic Linear Program
SIP	Stochastic Integer Program
et al.	et alii (and others)
etc.	et cetera
vs.	versus
a.k.a.	also known as
API	Application Programming Interface
RTS	runtime system
PPL	Parallel Programming Laboratory
DoD	Department of Defense
TACC	Tanker Airlift Control Center
NCSA	National Center for Supercomputing Applications

1 Introduction

Optimization problems are a very important class of problems in mathematical science. The entire area of operations research is dedicated to applying mathematical modeling, statistical analysis and mathematical optimization to complex decision making problems faced by organizations. Some of the decision problems can be formulated into a linear or an integer program. In its canonical form, a linear program can be expressed as

$$\begin{gathered}\text{maximize } c^T x \\ \text{subject to } Ax \leq b \\ \text{and } x \geq 0 \text{ for linear programs} \\ \text{or} x \in I \text{ for integer programs}\end{gathered}$$

The goal in linear/integer programs is to maximize the objective function given a set of constraints. However, most of the real world applications involve uncertainty, and therefore some of the data that form the constraints is unknown. Stochastic optimization is used for optimization under uncertainty: for example, allocating resources optimally when the demands are uncertain. It can be applied to those problems where the probabilistic distribution of the data is either known or can be estimated. It has applications in a broad range of areas, ranging from finance to transportation to energy optimization.

In the two-stage stochastic programming approach, the decision variables are divided into two sets - first stage variables and the second stage or recourse variables. First stage variables are those that have to be decided before the actual realization of the uncertain parameters becomes available. Subsequently, once the uncertain events have presented themselves, further design or operational policy improvements can be made by selecting at a certain cost, the values of the second stage or recourse variables [1]. A standard formulation of the two stage stochastic program is [2,3]:

$$\min_{x \in X} \{g(x) := c^T x + E[Q(x, \xi(\omega))]\}, \tag{1.1}$$

where,

$$Q(x, \xi) := \inf_{y \in Y} \{q^T y : Wy \geq h - Tx\} \tag{1.2}$$

is the optimal value and $\xi := (q, T, W, h)$ denotes the vector of parameters of the second stage problem. Problem (1.1) is the first stage in which vector x is obtained and it gives the lower bound on the optimal cost. Vector x is then used in Problem (1.2) to evaluate the actual costs of each scenario realizations. The expected cost $E[Q(x, \xi(\omega))]$ is obtained by solving large number of scenarios and then taking a weighted average of their costs according to the probability of their occurance. The weighted average gives the upper bound on the optimal value. Scenario optimization also provides recourse information (from the dual space) that can be added as constraints to the stage 1 problem to improve the vector x. This method is also called the multicut L-shaped algorithm proposed by Birge et.al. [4]. This cycle of a stage 1 solve followed by a stage 2 solve (also called an *iteration* or a *round*) is terminated when the lower bound and upper bound fall within the desired threshold i.e. when the solution with the expected accuracy has been found.

In this work, we study the DoD's airlift fleet assignment problem, which is a typical example of stochastic optimization. Various missions of the US Air Mobility Command (AMC) involve uncertainty based on the worldwide tension and commitments. DoD manages a fleet of over 1300 aircrafts [5] that operate globally, frequently with little notice, to destinations lacking infrastructures but populated with hostile forces. It operates in a uniquely challenging environment and is also the world's largest airline. Aircraft 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. 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. We model the allocation process as a two-stage stochastic mixed-integer program with complete recourse. Aircraft are allocated in the first stage; the second stage subproblems conduct more detailed planning with probability-weighted mission and cargo demands over hundreds of scenarios.

Aircraft allocation is a complex problem involving several thousands of variables and constraints. The number of scenarios faced by the problem can vary from hundreds to millions. It is important to generate good solutions within a reasonable amount of time. Since reallocations are costly, it is important to plan for uncertainties in the system. Serial computing techniques, the current state-of-the-practice in the DoD logistics decision- making environment, reach practical limits quickly, so contemporary decision support tools provide approximate solutions at best. In the last decade, there has been tremendous growth in the computing power of supercomputers. Nowadays, researchers have access to supercomputers that can provide many petaflops of peak performance. Stochastic integer optimizations have not yet exploited these large scale distributed memory machines because of the computational complexity and various research challenges that it poses. However, there is reasonable prior work on parallelization of stochastic linear programs. Jeff Linderoth, et.al. in their 2005 paper [6], discuss linear stochastic optimization algorithms and their implementation on the grid platform. They ran their program on 1300 machine grid and were able to solve an instance of storm [7] with $N = 10^7$ scenarios in 32 hours. The equivalent linear program has approximately 10^9 constraints and 10^{10} variables. A recent attempt at stochastic optimizations on current-scale supercomputers has been made by Lubin, et.al. [8]. They present a scalable parallel algorithm for the economic dispatch(ED) problem - a core part of energy grid optimizations. ED problem can be modeled into a two-stage quadratic stochastic optimization problem. They obtain 96% strong scaling efficiency at 32 racks (131,072 cores) of the IBM's Blue Gene/P system, on a problem with 4,700 variables in the first stage and 14,000 variables in the second stage scenarios. The largest linear system they solve directly is of order 1.4 billion.

On the other hand, stochastic integer optimizations are extremely hard problems because they require solving mixed-integer programs, which are NP-Hard problems. However, if successful, it has huge potential in areas such as engineering design, contingency planning in manufacturing [9], military operations planning, unit commitment problem in energy systems [10], and many more. Nwana et.al. [11] have discussed their implementation of a two-stage parallel branch and bound algorithm for mixed-integer programs. Shabbir et.al. [12] present a finite branch and bound algorithm for two-stage stochastic integer programs but do not make an attempt at parallelization of their algorithm. In this thesis we present our attempt to obtain provably optimal solutions by massive parallelism of exact methods for stochastic integer optimization. We explore the challenges in stochastic integer optimization and present a novel branch-and-bound based approach that is capable of scaling to hundreds of thousands of processors and thus making it possible to solve huge stochastic optimization problems that have remained unsolved in the past.

In Chapter 2, we discuss the US Air Mobility Command's aircraft allocation problem and our formulation of the stochastic model for this resource allocation problem.

In Chapter 3, we describe our implementation of the Bender's decomposition method for stochastic optimization using Charm++. Various novel performance-optimization techniques that led to significant improvement in the execution time have been discussed.

Chapter 4 discusses the scalability challenges in the Bender's method and how we propose to address them in our massively parallel branch-and-bound based implementation of the bender's decomposition method. We then present our branch-and-bound design. In Chapter 5, we present and analyze the performance of this approach. Conclusion and Future Work are discussed in Chapter 7.

2 The Airlift Fleet Assignment Problem

In this Chapter, we present here an analytical approach to replicate the TACCs planning cycle. The associated model is a stochastic mixed integer program that allocates aircraft to three of the primary mission types flown by the Air Mobility Command:

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

Aircraft are allocated by airlift wing, aircraft type, mission type, and day. The time horizons considered are between 5 and 30 days.

We model the problem as a two stage stochastic program with recourse. In the first stage, before a realization of the demands are known, one has to make a decision about the allocation of the aircrafts to different missions at each base location. At the second stage, after a realization of the demand becomes known, one may have to take the recourse action of increasing the capacity for satisfying an unmet demand. The expected cost of an allocation is computed by solving the second stage optimization problem for a large number of realizations. Finally, an allocation that minimizes the expected cost is computed. Complete model formulation is given in Appendix A.

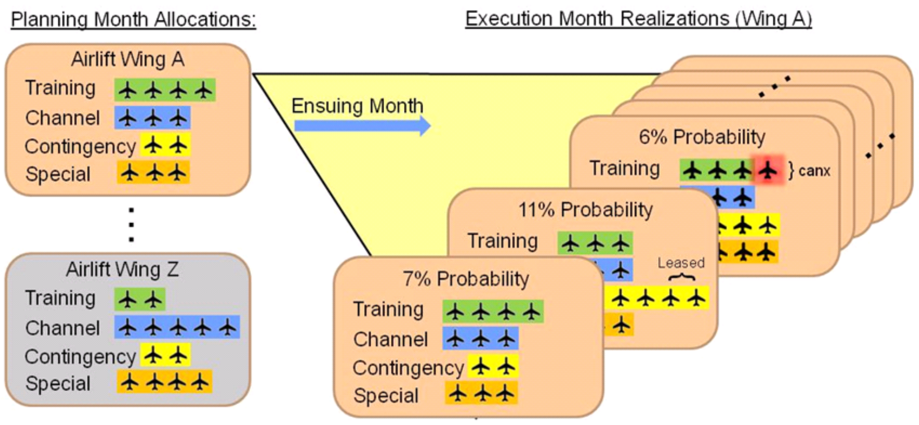


Figure 2.1: Tanker Airlift Control Center (TACC) allocations to wings incorporate a *best guess* of next month's requirements.

2.1 Model Evaluation

We conducted two experiments on 5t SIP model to quantify the cost benefits obtained from stochastic optimization compared to deterministic optimization. In each of these experiments, we evaluated each scenario with the optimal allocation obtained from the respective optimization and compared the cost difference. In Experiment 1 (Figure 2.2), 120 scenarios were generated with channel demand drawn from a normal distribution, SAAM demand following a Poisson distribution and the contingencies occurring randomly with some assigned probabilities of occurrence.

In Experiment 2 (Figure 2.3), 6 scenarios were generated with normally-distributed channel missions and randomly-distributed contingencies. However, SAAM demands were generated to create highly-stressed scenarios. Thus, SAAM demands were scaled by some factor in each scenario, so that they are very low, low, average, high or very high. This allows us to more clearly demonstrate the benefits of stochastic optimization when black swan events occur.

Figure 2.2 shows that up to 6% cost benefit (3% on average) can be obtained from stochastic optimization. Figure 2.3 shows that in extreme high-demand scenarios

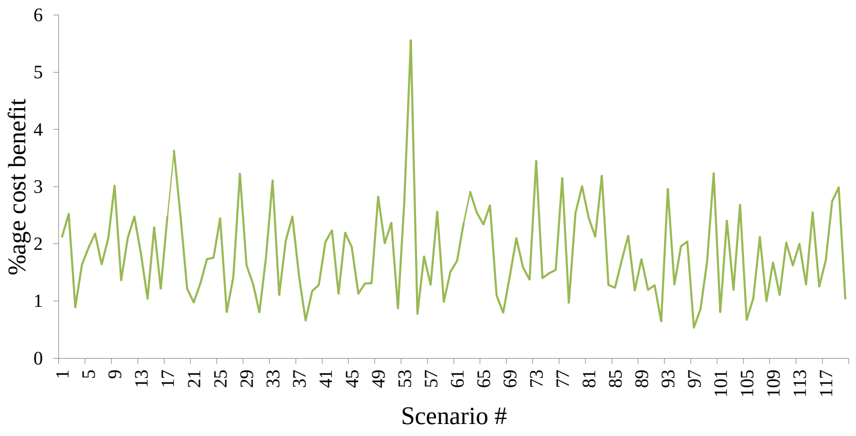


Figure 2.2: Hedging against future uncertainities can play a significant cost-saving role. In this simplified 6-scenario example, circle icons correspond to each scenarioś cost using deterministic optimization of expected demands; the horizontal average is depicted as a dashed line. In contrast, the squares correspond to stochastic optimization costs for each scenario (with the dashed-line average). Histobars depict the percentage cost savings of the stochastic optimization. Stochastic optimization yields only slightly more costly solutions when actual demands are low (scenarios 1,4, and 5), but much less costly when the demands are elevated(scenarios 2, 3, and 6).

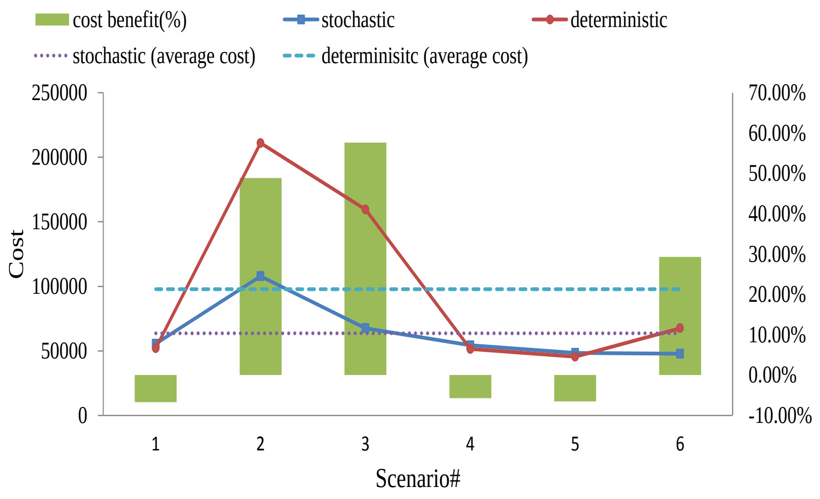


Figure 2.3: Improvement in cost using stochastic optimization with 6 scenarios

Table 2.1: Model Sizes of Interest (120 scenarios)

Model Name	Num Stg1 variables	Num Stg2 variables	Num Stg2 constraints
3t	255	1076400	668640
5t	345	1663440	1064280
10t	570	3068760	1988640
15t	795	4157040	2805000
30t	1470	7956480	5573400

stochastic optimization can reduce the operation costs by as much as 57% while incurring just 5% extra cost on the average scenarios. Stochastic optimization also reduces the variation in operating costs during different months of operations by as much as 66%.

For all of our experiments in subsequent chapters, we use a set of two-stage models of different sizes. These models formulate the airlift fleet assignment problem over different time horizons mainly 3, 5, 10, 15 and 30 days. More detailed descriptions of these models are given in Table 2.1.

For each of the models, there exist both SLP and SIP formulations. In SLP formulation, both stage 1 and stage 2 are linear programs, while in the SIP formulation, stage 1 is a mixed-integer program and stage 2 is a linear program.

3 Master Worker Parallelization

Multi-cut Bender's decomposition [13] is a popular technique used in solving mixed-integer stochastic programs. In the two-stage model formulation of the airlift fleet management, stage 1 is a mixed-integer program and stage 2 an integer program. However, we relax the stage 2 to a linear program because solving as many integer programs as there are scenarios is computationally expensive and will take an unfeasibly large amount of time even with the state-of-the-art algorithms on modern supercomputers. Linear programs in stage 2 will give us a bound on the actual costs of the scenario. The stochastic formulation described in Chapter 2 consists of coarse grained computations because the linear/integer programs in stage 1 and 2 cannot be broken down trivially. Linear/integer program optimizations are delegated to numeric libraries, and these optimizations form the fundamental grain of computation.

The Master-worker approach consists of the following steps:

1. Single stage 1 compute object (master) proposes new allocation
2. Collection of stage 2 compute objects (workers) evaluate this allocation and provide feedback to stage 1
3. Multiple-rounds of master-worker interaction takes place until allocation within desired accuracy is found

There are three different types of objects in our implementation:

1. The stage 1 solver

2. Communicator a.k.a Comm

3. Stage 2 solvers

Our implementation of the master worker parallelization is done in Charm++. Charm++ [14–17] is an object-oriented parallel programming framework. It includes a programming framework based on C++ and an adaptive run time system. Objects participating in the parallel control flow are called Chares and Chare Arrays. A Chare is the basic unit of computation in Charm++. It can be created on any processor and operated on remotely (through entry methods). A Chare Array is a collection of chares indexed by some index type. They can be considered similar to any other arrays we come across in programming languages. We will refer to them as chare arrays, Charm++ arrays or simply arrays interchangeably. Each element of a chare array is called an array element. In the master worker implementation, stage 1 solver and the *Comm* are chare objects and stage 2 solvers form a chare array. The specific advantage of using Charm++ as the parallel programming model is the ease of expressing the parallel program workflow. Additionally, Charm++ has a robust adaptive run time system capable of running applications on hundreds of thousands of processors. We can also use the fault tolerance capabilities of Charm++ to recover from failures when doing large scale runs.

Both the stage 1 solver and stage 2 solvers solve their respective LP/IP using a numeric library. For all our implementations, we use a commercially available solver for linear and mixed integer programming called the Gurobi optimizer [18]. The solve times of the LP/IP is variable and unpredictable as it depends on the nature of the problem. Additionally, the ”advanced start” feature (starting from the basis of the previous solve) provided by the solver significantly reduces the solve times on an average. However, if we use the ”advanced start” feature of the LP solver, the solution times are highly variable. In some cases, it is smaller than 100

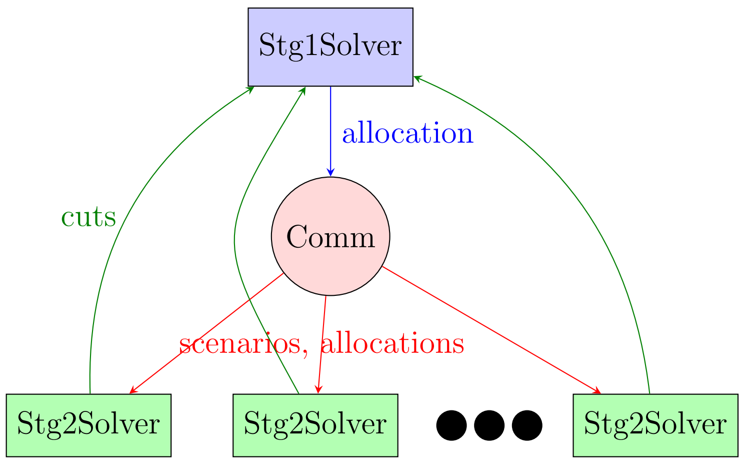


Figure 3.1: Design of the Master Slave Implementation. Single *Comm* object will suffice for small number of scenarios. Multiple *Comm* objects can be used when #scenarios are large.

milliseconds, while in others it is larger than 10 seconds. Therefore, there is a load imbalance because of variable, unpredictable compute grain sizes. To address this issue, we employed a work request mechanism instead of apriori load distribution. The Stage 1 solver sends the newly generated allocation to the *Communicator a.k.a. Comm* object. Stage 2 objects request work from *Comm* when idle. *Comm* balances load by responding to stage 2 worker requests in First-In-First-Out(FIFO) order. Therefore, stage 2 workers keep working until the allocation is evaluated under all scenarios. Note that *Comm* runs on a dedicated PE in order to keep it responsive to stage 2 worker requests.

The aircraft assignment problem that we are dealing with has 1500 integer variables in stage 1. Stage 1 being an mixed-integer program is a NP-Hard Problem. Sequential implementation of the stage 1 solve makes the problem intractable. To visualize the execution work flow of the program, we use the performance analysis tool in Charm++ called Projections [19]. Figure 3.2 demonstrates the stage 1 bottleneck through timeline view of projections. Each horizontal line corresponds to a processor. Yellow and green color represents the stage 1 and stage 2 solves respectively. As the timeline shows, all of the stage 2 solvers stay idle while the

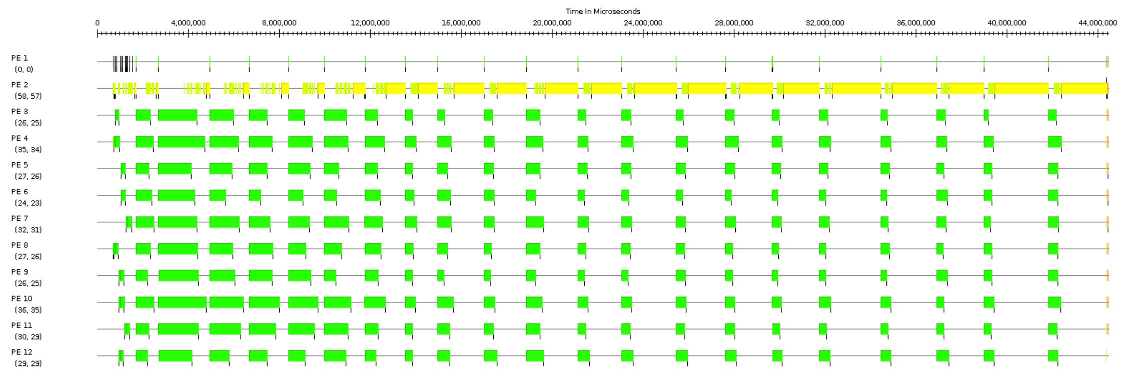


Figure 3.2: Projection traces of Master Worker Implementation from a 5t SIP Model Run

stage 1 solve is being performed. Additionally, the time taken by the stage 1 solve continues increasing with the number of iterations as stage 1 accumulates cuts from stage 2 (more details is Section 3.1). To make this problem tractable, we relax the integer variables in stage 1 and solve it as a linear program and call the model - a stochastic linear program (SLP). Later on, in Chapter 4 we discuss techniques to solve the stage 1 as a mixed-integer program i.e. the stochastic integer formulation, SIP.

Figure 3.2 shows the stage 1 and stage 2 solve times for a 5t SIP Model. The SLP model shows a similiar pattern except that the Stage 1 solve times are much smaller. As we can see that the stage 1 solve time shows an increasing trend which means increased serial bottleneck and therefore decreased parallel efficiency. In the following two sections we present two optimizations to reduce the stage 1 and stage 2 solve times. In Section 3.1, we present a cut management scheme that bounds the stage 1 solve times by containing the size of the stage 1 model and in Section 3.2, clustering of scenarios is proposed that reduces the average stage 2 solve time.

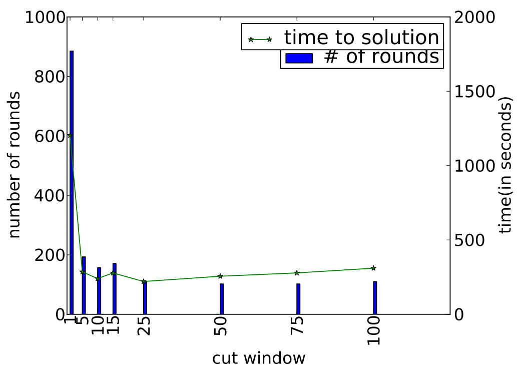
3.1 Cut Management

In each iteration of the two-stage Multi-cut Benderś decomposition method, #scenario cuts are added to the stage 1 model. Therefore, size of the stage 1 model increases linearly with the iteration number. This linear increase in model size leads to an increase in the solve time of the model(as is evident from Figure 3.2). Increase in the model size also increases the memory requirements.

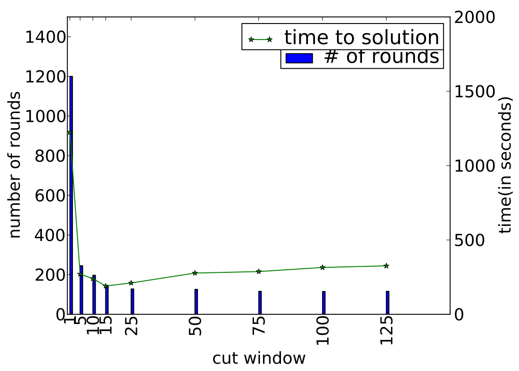
In order to contain the size of the stage 1 model, we put a user defined limit on the number of cuts that can stored in the stage 1 model. Cuts are assigned scores based on the rate of their usage (Equation (3.1)).

$$\text{cut\_score} = \frac{\text{number of rounds in which it is used}}{\text{total number of rounds since its generation}} \tag{3.1}$$

The Gurobi optimizer provides the slack value of each constraint which tells us whether that cut was active during the last optimization or not. 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. After every stage 1 solve, score of each of the cuts is updated based on their activity/inactivity in the last solve. We put a user defined ceiling on the the number of cuts to store in stage 1. Whenever the number of cuts exceed this threshold (we call this threshold *cut-window*), the lowest scoring cuts are discarded. Discarding cuts does not affect the solution to the problem because any important cut that might get discarded will always be regenerated in the subsequent iterations. This can lead to increase in the number of iterations required to converge. However, our experiments show that the benefits of this approach far outweigh potential disadvantages. Bounded stage 1 model size significantly reduces the stage 1 solve time. It also flattens the increasing memory usage for the stage 1 gurobi model. Our Experiments show (Figure 3.4) that even though the number of iterations required to reach to convergence increases, the



(a) 5 time period model (solved to 0.1% convergence on 8 cores of 2.26 GHz Dual Nehalem)



(b) 10 time period model (solved to 1% convergence on 32 cores of 2.67 GHz Intel Xeon hex-core processors)

Figure 3.3: Performance of 5 and 10 time period models with different *Cut Windows*

total time to convergence can reduce significantly because of the smaller stage 1 solve times. Moreover, choosing the *cut-window* influences program behavior significantly. Too small *cut-window* may lead to no or slow convergence and too high *cut-window* will cause the stage 1 memory usage and solve times to become very large. To compare different cut-windows we performed an experiment that measures the time to solution of 10t SLP model with 120 scenarios on 32 cores. The plot in Figure 3.3 shows that even though the number of rounds required to converge are more for smaller *cut-windows*, total time to solution is much less.

As a further optimization, we try to minimize the effect of discarding cuts that could potentially be useful. Along with the cuts for each scenario we also add a surrogate constraint to the stage 1 model after each iteration. A *surrogate constraint* [20] is obtained by aggregating all the cuts obtained from the scenario subproblems. This method of adding only a single constraint (derived from the stage 2 scenario subproblems) to the stage-1 model is also called L-shaped algorithm for two-stage stochastic programs [21]. Adding surrogate constraint offers two benefits:

- It contains some information from each of the parent cuts and thus minimize the loss of information when some of the parent cuts are discarded.

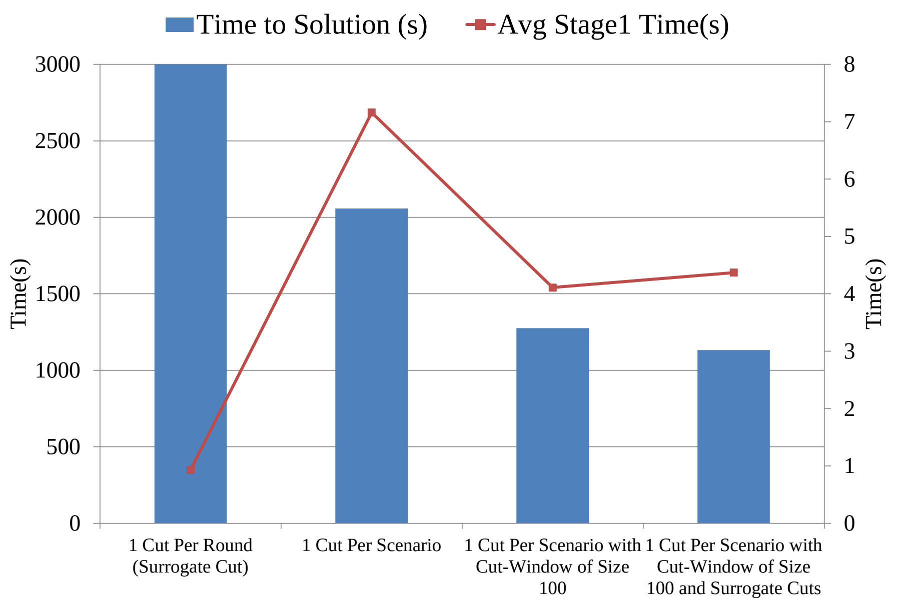


Figure 3.4: Time to solution for different cut collection schemes used in stage 1

- It can also capture useful information that cannot be extracted from the parent constraints individually but is nevertheless a consequence of their conjunction.

Figure 3.4 shows a plot comparing different strategies namely, L-shaped, Multi-cut L-shaped and hybrid of these two. L-shaped algorithm (bar 1) takes magnitude of order more time when compared to the other approaches. As observed in Figure 3.3 the multicut L-shaped algorithm with cut-retirement (bar 3) performs better than the multicut shaped algorithm without cut-retirement (bar 2). Using surrogate constraint along with multicut L-shaped algorithm and cut-retirement gives the best performance (bar 4).

3.2 Scenario Clustering

Advanced start (or warm start) feature in linear programming solvers can lead to significant reduction in solve times when similar problems are solved consecutively. In advanced start, the solver begins with the basis of the previous solve for the next optimization. We exploit this feature to speedup the stage 2 solves. Since, number of

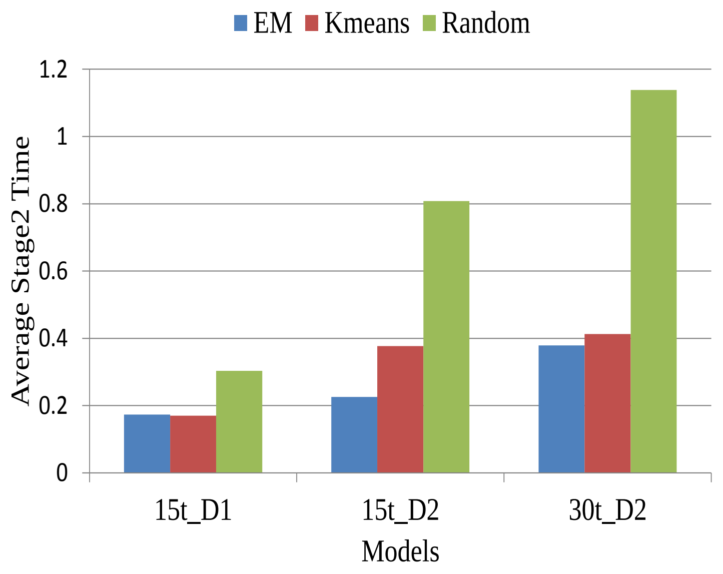
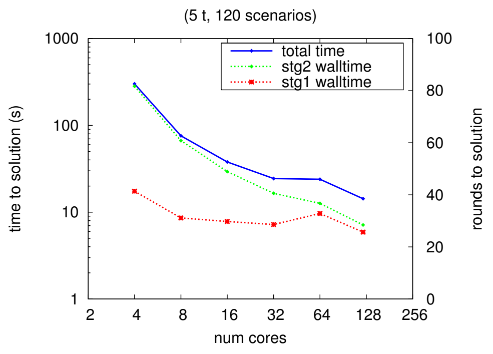
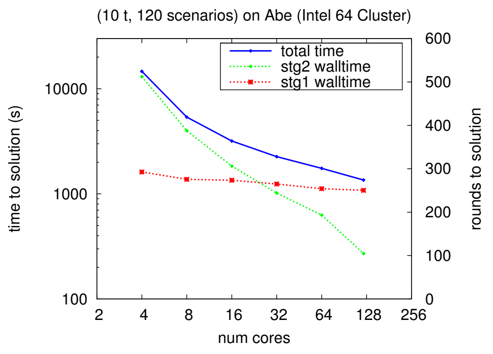


Figure 3.5: Comparing reduction in average stage 2 solve time using different clustering mechanisms. Scenarios are clustered based on similarity in demands.

scenarios for real-world stochastic optimization problems are very large in number, even small amount of reduction in the average stage 2 solve time can have sizeable payoffs. In our formulation of the airlift fleet assignment problem, scenarios differ only in the right hand sides (RHS) of the constraints. Constraints in stage 2 are of two types - demand and allocation constraints. We cluster the scenarios into K clusters based on the demands of the scenarios. K is the number of available stage 2 solvers. Each cluster is assigned to a specific processor. In Figure 3.5 we compare two clustering mechanisms, namely Expectation Maximization (EM) [22] and KMeans [23], with random distribution of scenarios. The timings are compared on three different demand sets namely D1 for 15t Model, D2 for 15t and 30t Model. In demand set D1, demands vary randomly between X and $X + 10$ while in demand set D2 demands are spaced evenly between X to $X * 5$. In all of the three cases, EM clustering algorithm gives greater than 62% reduction in average stage 2 solve times.



(a) 5 time period problem solved to 0.1% convergence



(b) 10 time period problem solved to 0.1% convergence

Figure 3.6: Scalability plots for 5 and 10 time period models

3.3 Performance

In this section, we present the scalability results of our optimized implementation of master-worker parallelization for two-stage stochastic optimization of airlift fleet assignment problem. Figure 3.6 shows scalability of the $5t$ and $10t$ SLP model with 120 scenarios run on a cluster with Intel x86-64 processors. Blue line corresponds to the total execution time while the green and red lines are the times spent in Stage 1 and Stage 2 solves respectively. Total execution time is the sum of the times spent in Stage 1 and 2. Stage 1 being a sequential bottleneck does not show any scaling with the increase in number of processors. Stage 2, on the other hand, shows perfect scaling up to 64 processors but the curve flattens beyond 64 cores because of the sequential nature of individual stage 2 solves. Figure 3.6 also demonstrates the Amdahl's law [24] which puts a limit on the maximum speedup that can be obtained from the master worker implementation.

4 Parallel Branch and Bound

For SIP models, each stage 1 problem is to be solved as an integer program with a master-worker parallel structure described in Chapter 3. Such parallelization successfully reduces the time taken to evaluate an allocation across all scenarios. However, this makes the serialization during the master phase (stage 1 IP) increasingly prominent as we scale to more processors. Problems that involve allocation decisions for multiple time periods (10t, 15t, 30t) are significantly larger and intractable with this approach. Such problems require many more rounds to converge, and also large collection of cuts in stage 1. Larger stage 1 size just inflates the sequential integer optimization time. The stage 1 IP is a significant serial bottleneck that impacts the scalability of the problem. The solution of integer programs in each round has the further disadvantage that each round repeats the enumerative search inherent in a branch-and-bound strategy. To overcome this redundant and repetitive search, we have changed the solution strategy.

To address the serial bottleneck described above, the problem is initially relaxed to form a stochastic linear program (SLP) by linearizing the stage 1 integer variables. The SLP requires significantly less time than the IP form, thereby reducing the stage 1 serial bottleneck. This relaxed SLP is solved at the root vertex and then a single branch and bound enumerative search is conducted. At each vertex of this branch-and-bound tree we solve a stochastic linear program. This strategy has several advantages. First, we eliminate the repetitive enumeration inherent in the previous approach. Second, we use the fact that cuts generated in one part of the tree are valid at all vertices of the search tree. Thus at each vertex of the tree, there is a

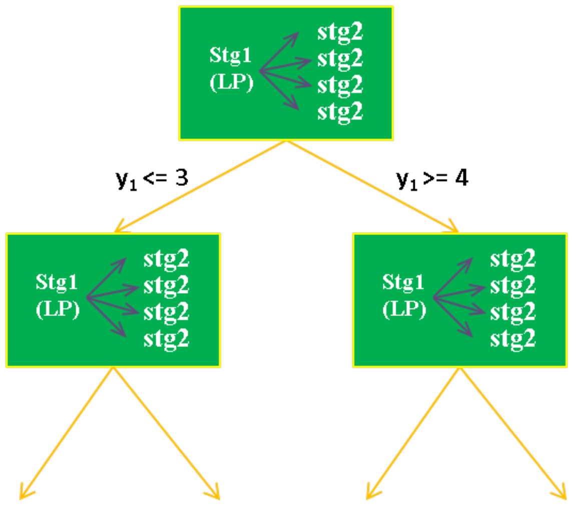


Figure 4.1: Two sources of parallelism apply to this research. Within each branch-and-bound vertex (box) a stochastic program is parallelized using LP relaxation in stage 1. In turn, each vertex of the resulting branch-and-bound tree can be parallelized to converge toward the stage 1 integer solution.

sufficiently rich set of valid cuts which can be used to establish an initial bound at that vertex.

Because we now control the branch and bound process (in contrast with the previous approach of allowing Gurobi to solve the integer programs) we have more opportunity to parallelize computation and effectively use a larger number of processors. Thus there are two available sources of parallelism: simultaneous optimization of vertices (generated by parallel branching) in the branch-and-bound tree and scenario parallelization within each vertex.

Figure 4.1 shows a schematic of a branch-and-bound tree. Notice that the root vertex is an SLP and can be solved using the same ideas as in Chapter 3. However, stage 1 is a linear program and thus requires significantly less time than an integer program thereby reducing the stage 1 serial bottleneck. Once the root vertex SLP converges to a desired level of accuracy, a branch-and-bound tree is formed by branching on suitable fractional variables in the root vertex SLP. At this stage a second level of parallelism is possible. By successively branching on several variables it is possible to enlarge the tree to create a number of search vertices within it. These vertices can now be solved in parallel. While this is a relatively apparent parallelism, there is a chance that prematurely branching on variables can lead to the exploration of parts of the search tree that could otherwise have been pruned due to bounds. This requires a judicious use of premature branching. We now discuss the various considerations and experience with the master slave implementation that guided our design for the branch-and-bound approach.

4.1 Design Considerations

- Solve time variability. Unlike most iterative, scientific applications, linear programs result in highly variable solve times. The solve times are also largely unpredictable. Load balancing schemes that depend on the principle of persistence [25] may not function very well in this context. This implies that our parallel design cannot decompose and distribute the computational load a priori. Instead, we have to rely on mechanisms that assign computational tasks to processors that become idle first. This suggests a pull-based scheme to avoid idle time and achieve load balance; with tasks potentially passing through queues to ensure appropriate fairness and prioritization policies.

- Memory limits. The amount of memory required to represent a reasonable stage 1 model in a form that can be consumed by the LP library is quite high, and has grown further as we incorporate multiple time periods for allocation. Consequently, this forces us to refrain from creating a new stage 1 instance of the problem for each vertex that is created via a branching decision in the branch-and-bound process. The number of stage 1 model instances is hence O(n), where n is the number of processors and not O(v), where v is the number of vertices generated in the branch-and-bound tree.

- Cut Sharing. Despite the memory limits that restrict the number of model instances which can be created, the ability to share cuts across vertices in the tree means that we can use the same model instance for multiple branch-and-bound vertices. Hence we have the flexibility to explore and solve multiple different branch-and-bound vertices concurrently. This allows our parallel structure to describe and manipulate the computations in terms of the natural unit: the branch-and-bound vertex; while managing cuts and memory at a processor-level.

- Subtree Locality. As mentioned earlier in this section, judicious management of the collection of cuts becomes important for reining in the stage 1 solve times. Although theory allows us to generate cuts based on a single branch-and-bound vertex and share it with all other vertices, we refrain from doing so to avoid the memory bloat that will accompany such a policy. Hence, our design should have the capacity to switch between broadcasting generated cuts to the whole branch-and-bound tree or sharing them only within the neighborhood (in the branch-and-bound tree) of the vertex that generated that cut. The natural "neighborhood" of a vertex in the branch-and-bound tree is the model instance that it shares with other branch-and-bound vertices.

- Migration costs. The cost of migrating a stage 1 model instance between processors is high because of its large size. As a partial remedy, we frequently choose to keep generated cuts local to a subtree, so the multiple stage 1 models may slowly specialize cut collection to their associated subtrees. In such a context, migrating vertices across processors should be only be a secondary load-balancing mechanism as it either involves high data movement costs or discards valuable vertex-specific cut information. The parallel design should hence include capabilities for avoiding idle time without moving branch-and-bound vertices around

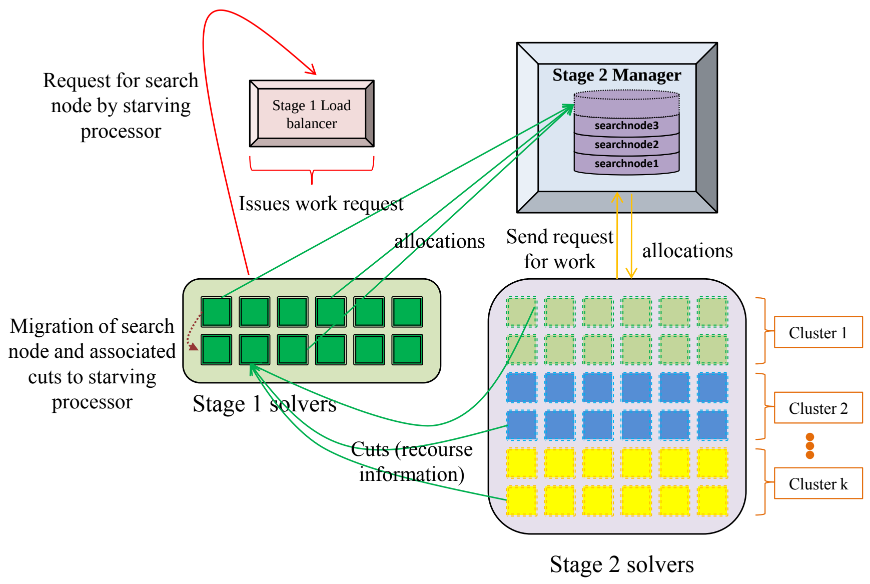


Figure 4.2: Multiple levels of parallelization requires a specialized architecture. Each stage has a balancer to keep processors busy, and a bank of processors which in case of stage 2 are grouped by cargo-demand pattern.

across stage 1 model instances. This suggests a two (multi) tier load balancing approach.

- Parallel Branch and Bound Architecture. The complexity inherent in our multiple-parallelization scheme drives a need for a specialized computing architecture; Figure 4.2 illustrates this design. Available processors are divided into four different entities: one stage 1 load balancer (S1LB), one stage 2 manager (S2M) and rest of the processors act either in the capacity of a stage 1 solver or a stage 2 solver.

  Each stage 1 solver is assigned a set of search vertices and is responsible for performing stage 1 optimizations when stage 2 cuts become available for the specific vertex. The S1LB coordinates with the stage 1 solvers to provide work to idle stage 1 solvers, and maintains a priority queue to prefer high priority vertices for optimization. The S1LB also tracks the load (the number of queued vertices) at each stage 1 solver. Each stage 1 solver dynamically updates S1LB with its load status whenever there is a change in its load. Whenever the load at a stage 1 solver becomes zero, the S1LB issues a work request to the solver with the highest load, which then relinquishes half of its queued load to the starving stage 1 solver.

The stage 2 solvers process stage 2 of the stochastic optimization. Each stage 2 solver receives a scenario and an allocation from the S2M. It optimizes the stage 2 problem for the given scenario and allocation and returns the resulting cut to the appropriate stage 1 solver. To ensure that the memory resident advanced basis is beneficially used, each stage 2 solver is assigned a cluster of scenarios (clustered by cargo demand similarity).

The S2M maintains priority queues of allocations received from stage 1 solvers. When requested for work from stage 2 solvers, it fairly distributes the queued allocations by cycling over the array of stage 1 queues.

Figure 4.3 shows an instance of the branch-and-bound tree during the program execution. Colors in the background correspond to the processor on which the respective search vertices were explored. Each of the four colors in the background correspond to the four stage 1 solvers/processors. Light green color corresponds to the search vertices for whom stage 1 is being solved. Red colored vertices have been pruned because of infeasibility. Orange colored vertices are the incumbents and hence are also the leaf nodes. Vertices ready for the stage 1 solve are colored dark green while the vertices waiting for the stage 2 solve to return are colored in light blue. Data used to generate figure was dumped during the program execution and rendered using graphViz [26] and gvmap [27] softwares.

Figure 4.4 shows another instance of the branch-and-bound tree with the search vertices colored according to their lower bounds. Color of the vertices vary from dark blue to red, with blue color vertices having smaller lower bounds and red colored vertices having higher lower bounds. Nodes represented using concentric circles are the ones currently being explored while the ones with single circle have either been

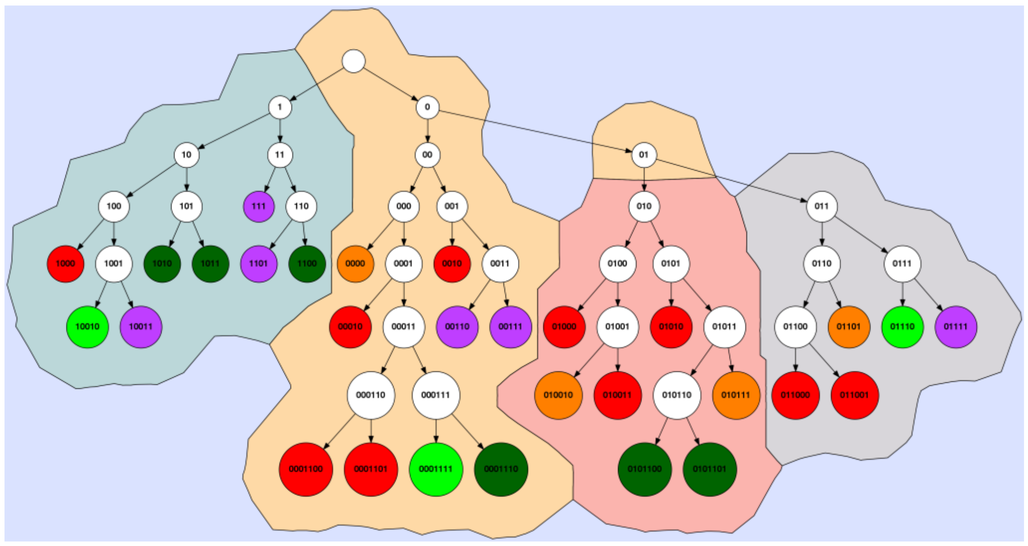


Figure 4.3: An instance of the branch-and-bound tree showing different states of the search vertices.

pruned, branched upon or are infeasible. Vertices that gave incumbents are shown as inverted triangles.

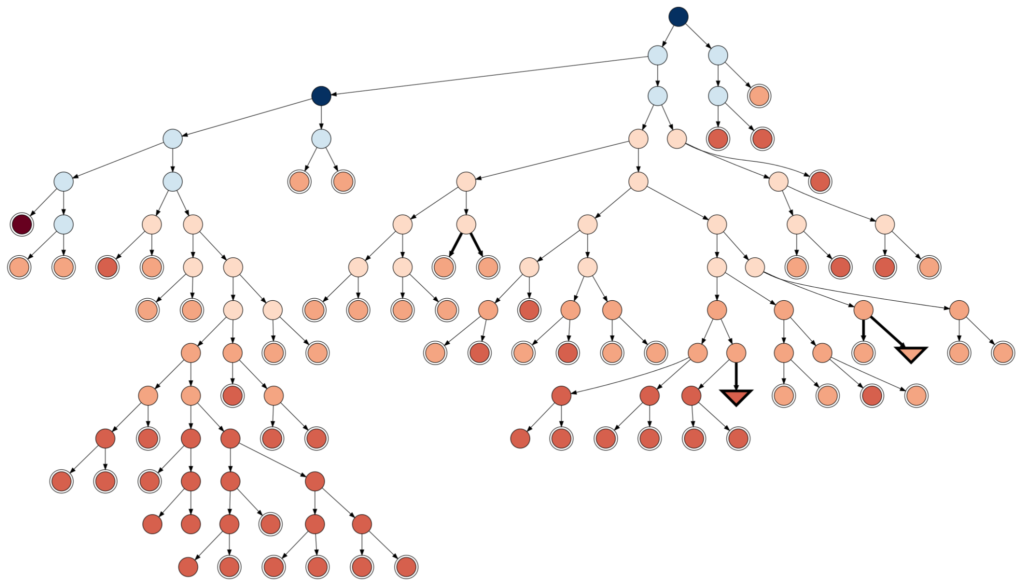


Figure 4.4: Another instance of the branch-and-bound tree with nodes colored on the basis of lower bounds at the vertices.

5 Computational Results

We first compare the projections timeline for the original master-slave implementation with the new branch-and-bound implementation as shown in Figure 5.1. The timeline on the right represents the execution traces of the master-slave version of the code: yellow bars represent the stage 1 solves; green represent stage 2 solves. The large gaps between the green bars in the right timeline indicate excessive idle time for stage 2 processors. In contrast, the timeline on the left represents the branch-and-bound version. This execution has four stage 1 solvers (represented in yellow) that are simultaneously exploring the branch-and-bound tree for an optimal integer allocation. Note that the processors solving the stage 2 LPs have much smaller idle time gaps because multiple branch-and-bound vertices are concurrently being explored. This is true even if there is only a single processor exploring the branch-and-bound search tree. Consequently, system utilization is much higher.

Migration from a two-stage master-slave parallel implementation to a parallel branch-and-bound implementation allows us to harness more compute resources. In Figure 5.2 we plot the time to complete the branch-and-bound search for an optimum integer solution to the 5t SIP Model with 120 scenarios in stage 2. In this experiment, we use a single processor to explore the branch-and-bound tree while the other processors are devoted to solving LPs for the stage 2 scenarios. We also plot the time taken to solve and branch on the root vertex in the branch-and-bound tree. This curve is similar to an asymptotic minimum for the total execution time of the branch-and-bound exploration and represents the limit to which performance of the branch-and-bound search can be improved. We also observe that the root vertex

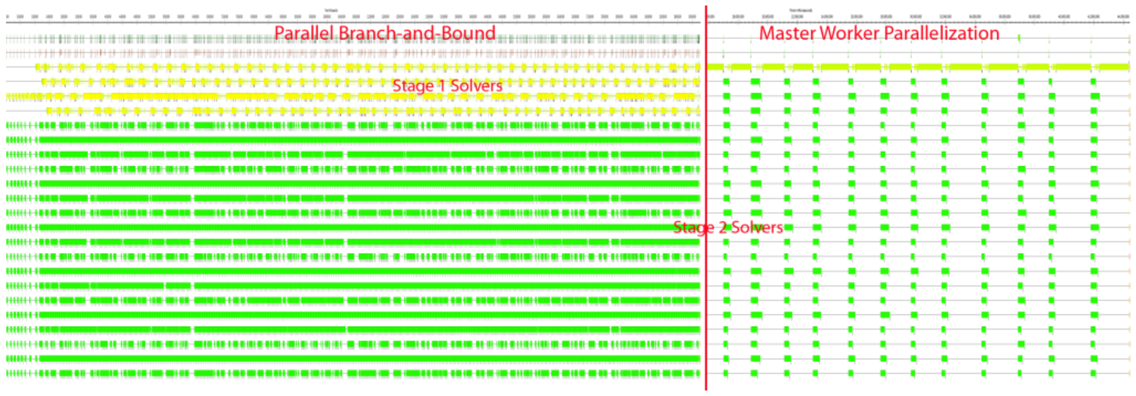


Figure 5.1: Projections traces compare processor utilization of the branch-and-bound and master-worker implementations (Chapter 3). Thick colored bars of each horizontal trace represent individual processor activity across time; thin gray lines indicate idle processors. The figure shows that processors are used more efficiently in the parallel branch-and-bound schema.

phase, which is akin to the original master-slave parallelization, starts degrading in performance as we add more processors. A possible cause is the communication bottlenecks that arise when many stage 2 solvers request work from a single stage 2 manager. This has possible fixes and we are currently evaluating these.

The next observation from Figure 5.2 is that the branch-and-bound implementation exhibits strong scaling i.e. improved performance when more processors are provided. This holds true to at least as many processors as the original master-slave implementation. In this simpler experiment where there is only one processor devoted to exploring the branch-and-bound tree, scalability is predominantly influenced by how well the processors can be utilized to solve the stage 2 LPs. Given that there are multiple branch-and-bound vertices, each of which require solving solving a two-stage Stochastic Linear Program (SLP), there is significantly more computational work. This can easily be parallelized over larger numbers of processors and results in enhanced scalability.

The final observation is the variation in the time required to solve the same problem for any given number of processors. This variation is quite sizeable, pos-

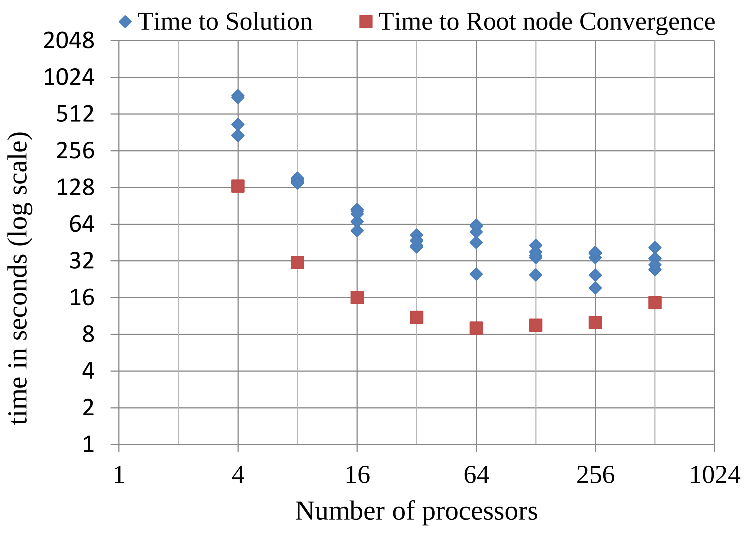


Figure 5.2: Solve times using branch-and-bound show good scalability. In this case a single processor is devoted to stage 1.

sibly above the level that is typically expected from using a shared hardware resource. There do not seem to be any apparent trends in this variation of execution times. It is worthwhile to note that the variation is introduced primarily during the branch-and-bound exploration phase because the time taken to solve the root vertex (represented by the lower curve) is quite consistent across multiple runs on a given number of processors. Our current explanation for this variation is that fluctuations in message transmission times affects both the order in which the tree exploration proceeds, as well as the solution obtained at each branch-and-bound vertex. Since each branch-and-bound vertex is solved to some convergence tolerance, the order in which the cuts are received can affect the solution to the LP problem. This in turn affects the order in which vertices in the branch-and-bound tree are explored. We are currently exploring strategies that mitigate these variations without sacrificing solution speed.

With an increase in the total number of processors, the number of stage 1 solvers can also be increased to allow parallel exploration of the branch-and-bound tree. Figure 5.3 shows scalability plot of 5t SIP Model(120 scenarios) with 1:4 ratio of stage 1 to stage 2 processors. Contrary to expectation, the allocation of more stage 1

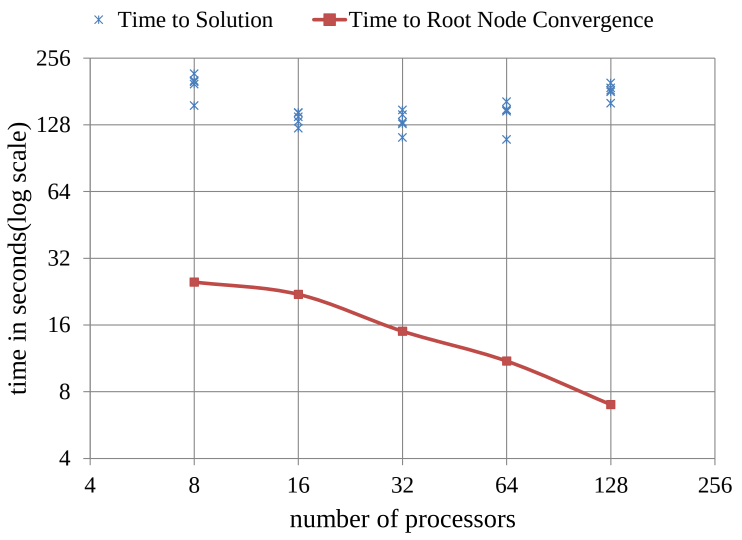


Figure 5.3: Proportionately increasing the stage 1 solvers with the number of processors can result in diminishing returns

processors does not always lead to reduced solve times. The plot also shows that the time taken to solve and branch on the root vertex is steadily decreasing with more processors. However, the branch-and-bound phase of the execution starts taking more time. We believe this is largely because of the lack of global monitoring and steering mechanisms that can prioritize the exploration of the whole search tree. To keep the increased number of stage 1 solvers busy, we either prematurely create more branch-and-bound vertices or get some vertices from other stage 1 solvers' queues. This in turn creates many more stage 2 problems. Thus, despite having more stage 2 solvers, we create a stage 2 bottleneck which also contributes to the increasing solution time.

Figures 5.4 and 5.5 further demonstrate how the ratio of stage 1 and stage 2 solvers can affect the program execution behavior. They represent a slice of the execution traces from two different runs of the branch-and-bound implementation. Only a representative slice of the processors is shown for a small portion of the total execution duration. The processors with execution traces in green are dedicated to solving the stage 2 LPs while the other processors are dedicated to exploring the branch-and-bound tree. Figure 5.4 shows that poor distribution of processors

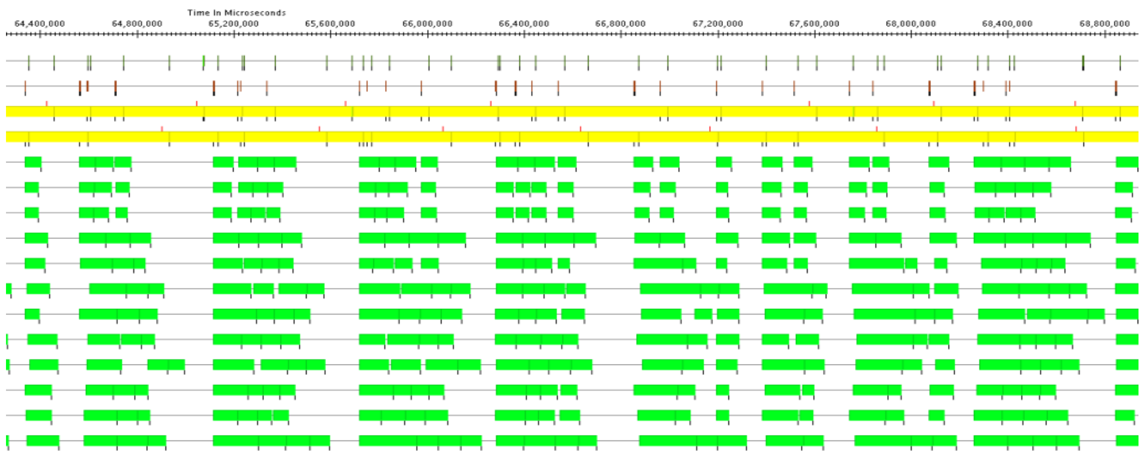


Figure 5.4: Idle Stage 2 Solvers with less than 50% processor utilization in some intervals. Yellow bars are the stage 1 solvers doing stage 1 solves and green color bars are the stage 2 solves on stage 2 solvers

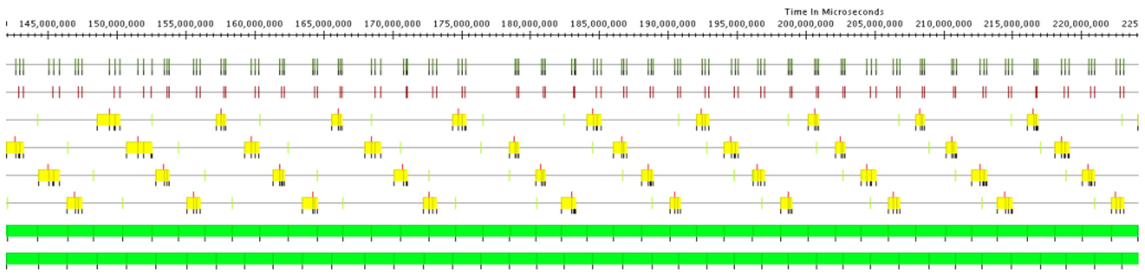


Figure 5.5: Overloaded Stage 2 Solvers. Yellow bars are the stage 1 solvers doing stage 1 solves and green color bars are the stage 2 solves on stage 2 solvers

can lead to under-utilized processors. Although not immediately apparent from the timeline, the stage 2 processor utilization in this time range is less than 50%! During those intervals, increasing the number of stage 1 solvers can increase the number of branch-and-bound tree vertices being explored simultaneously. On the other hand, too many stage 1 solvers can overwhelm the limited stage 2 solvers (Figure 5.5). The increased stage 2 response time in this case, can slow-down convergence of the tree vertices and also create unnecessary competition between high-priority vertices from one stage 1 solver with the low-priority vertices from other stage 1 solvers for the limited stage 2 solvers.

This calls for a judicious distribution of processors to stage 1 and stage 2 solvers. In the scalability plot in Figure 5.6, there are 4 stage 1 solvers for processor count up to 128 and 8 stage 1 solvers at processor count 256 and 512. Time on the y-axis is the time to solution after the root vertex convergence. We also plot the times when using a single stage 1 solver. Each data point plotted is the best performance that was obtained from several identically configured trials. We observe a relatively well-behaved speedup envelope where we can obtain better performance at larger processor counts by gradually adding more stage 1 solvers. This plot further clarifies the need to dynamically tune the proportions of stage 1 and stage 2 processors to obtain the best possible performance given any problem and number of processors. Its also worthwhile to note that using more processors beyond 512 will not result in further significant speedup with our current parallel implementation. This is because the root vertex SLP alone takes 14 seconds to solve. This limits the speedup we can achieve by parallel exploration of the branch-and-bound tree without addressing some of the challenges we have uncovered in the course of this work.

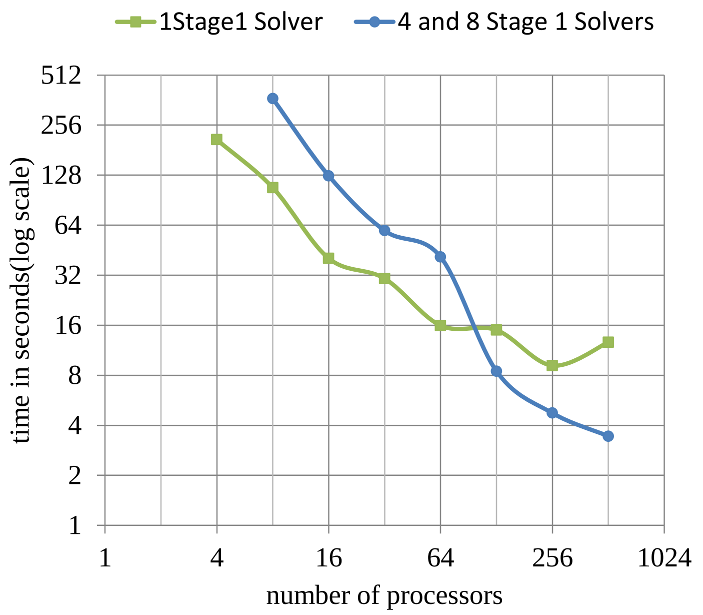


Figure 5.6: A judicious distribution of processors to stage 1 and stage 2 gives good speedup. Note that in this plot, time on the y-axis is the time to solution after the root node has converged.

Computational results for our branch-and-bound implementation are very encouraging. However, there are interesting issues raised by the large variability in the solve times and the need for optimally allocating processors. Our continuing efforts are targeted towards reducing variability without speedup penalty and for improved scalability to even more processors.

6 Conclusion and Future Work

We first presented our master-worker implementation for two-stage stochastic optimizations. We discussed two performance optimizations namely, cut retirement and scenario clustering. Cut retirement led to more than 50% reduction in the total execution time and scenario clustering in stage 2 reduced the stage 2 solve times by more than 60%. These strategies play a role in keeping the solution time of stochastic linear programs within practically acceptable time limits. However, these stratgies are not sufficient for driving stochastic integer programs to solution within desirable time limits. A master-worker implementation suffers from decreasing parallel efficiency with increase in the number of procesors and the speedup flattens out at modest number of processors. Nevertheless, this approach guided our design of a branch-and-bound based massively parallel approach that is capable of scaling to modern supercomputers having hundreds of thousands of processors. Our preliminary results show this capability through a scalability plot that goes upto 512 processors. At 512 processors, we obtain a speedup of 50 over 4 processor runs, which is a very promising result. Our results in the thesis were on the aircraft allocation problem but the presented approach is applicable to other two-stage stochastic programs that have integer variables in stage 1.

To the best of our knowledge, our work represents the first study of the behavior of branch-and-bound approaches for stochastic integer optimizations at this scale. Many interesting issues have been brought out e.g. reducing the variability in execution times of identical runs without suffering from speedup penalties. Obtaining solutions quickly also requires good steering strategy for exploring the branch-and-bound tree. These are challenging problems and will be addressed in our future work.

References

[1] Shabbir Ahmed, Alexander Shapiro, and Er Shapiro. The sample average approximation method for stochastic programs with integer recourse. *SIAM Journal of Optimization*, 12:479–502, 2002.

[2] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming.* Springer, New York, 1997.

[3] P.Kall and S.W.Wallace. *Stochastic Programming.* John Wiley and Sons, Chichester, England, 1994.

[4] J.R. Birge and F.V. Louveaux. A multicut algorithm for two-stage stochastic linear programs. *European Journal of Operational Research*, 34(3):384–392, 1988.

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

[6] J. Linderoth and S.J. Wright. Computational Grids for Stochastic Programming. *Applications of stochastic programming*, 5:61–77, 2005.

[7] J.M. Mulvey and A. Ruszczyński. A new scenario decomposition method for large-scale stochastic optimization. *Operations research*, pages 477–490, 1995.

[8] Miles Lubin, Cosmin G. Petra, Mihai Anitescu, and Victor 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.

[9] S. Ahmed, A.J. King, and G. Parija. A multi-stage stochastic integer programming approach for capacity expansion under uncertainty. *Journal of Global Optimization*, 26(1):3–24, 2003.

[10] E.M. Constantinescu, V.M. Zavala, M. Rocklin, S. Lee, and M. Anitescu. A computational framework for uncertainty quantification and stochastic optimization in unit commitment with wind power generation. *Power Systems, IEEE Transactions on*, (99):1–1, 2011.

[11] V. Nwana, K. Darby-Dowman, and G. Mitra. A two-stage parallel branch and bound algorithm for mixed integer programs. *IMA Journal of Management Mathematics*, 15(3):227–242, 2004.

[12] S. Ahmed, M. Tawarmalani, and N.V. Sahinidis. A Finite Branch-and-Bound Algorithm for Two-stage Stochastic Integer Programs. *Mathematical Programming*, 100(2):355–377, 2004.

[13] J. Benders. Partitioning procedures for solving mixed variables programming problems. *Numerische Mathematik 4*, pages 238–252, 1962.

[14] 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.

[15] 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.

[16] Laxmikant V. Kale, Eric Bohm, Celso L. Mendes, Terry Wilmarth, and Gengbin 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.

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

[18] Gurobi Optimization Inc. Gurobi Optimizer. Software, 2012. http://www.gurobi.com/welcome.html.

[19] Laxmikant V. Kale, Gengbin Zheng, Chee Wai Lee, and Sameer Kumar. Scaling applications to massively parallel machines using projections performance analysis tool. In *Future Generation Computer Systems Special Issue on: Large-Scale System Performance Modeling and Analysis*, volume 22, pages 347–358, February 2006.

[20] F. Glover. Surrogate constraints. *Operations Research*, pages 741–749, 1968.

[21] J.R. Birge. Decomposition and partitioning methods for multistage stochastic linear programs. *Operations Research*, pages 989–1007, 1985.

[22] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

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

[24] G.M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

[25] Laxmikant V. Kale. Some Essential Techniques for Developing Efficient Petascale Applications. July 2008.

[26] J. Ellson, E. Gansner, L. Koutsofios, S. North, and G. Woodhull. Graphvizopen source graph drawing tools. In *Graph Drawing*, pages 594–597. Springer, 2002.

[27] E.R. Gansner, Y. Hu, and S. Kobourov. Gmap: Visualizing graphs and clusters as maps. In *Pacific Visualization Symposium (PacificVis), 2010 IEEE*, pages 201–208. IEEE, 2010.

Appendix A: Stochastic Model Formulation of the Airlift Fleet Assignment Problem

The objective is to allocate the aircrafts to the different missions at the base locations. We model the aircraft allocation at the bases taking into account the average pair-wise demand between the locations. An aircraft is assigned a route, on which it serves a source destination pair (possibly in both directions). In each visit, the aircraft can potentially drop or pick-up some cargo, passengers and get fuel. The overall problem is decomposed into two stages as explained above.

Note that these allocations must be made without knowledge of the specific realizations. We have formulated the model to account for the various constraints imposed by different mission requirements which we describe shortly. It is also possible to include the rented aircrafts in the model and try to minimize the cost. The model is rich enough to account for the fuel consumption as the aircraft visits different cities.

Mission Types

- **Channel**

  These aircrafts originate and terminate at home base, making several enroute stops. They pickup and drop off cargo and passengers at major aerial ports. The routes are pre-planned based on forecast cargo demand patterns. A realization consists of random cargo and passenger draws for each of 30 days along each route (outbound and inbound). The routes are fixed, but the frequency and aircraft used may be varied (for the purposes of this model). The objective is to minimize the sum of late and undelivered cargo and passengers. Non-delivery penalties occur if cargo is undelivered for more than 7 days.

- **Special Assignment Airlift (SAAM) Missions**

  An aircraft is chartered for a specific time frame and originates from home station. A realization consists of random aircraft required, aircraft type, mission type, mission duration, and flying hour draws for each of 30 days. Demand is aircraft centric not cargo centric, and is of moderate variance. There are opportunities for some SAAM missions to carry channel cargo. The unmet missions are penalized at the rate of the associated (high penalty) aircraft.

- **Contingency**

  These are similar to channel missions in that they require cargo and passengers to be carried from one location to other. However, they differ from channel missions due to high demand variance and localized destinations. A realization consists of a Bernoulli draw for each type of Contingency. There are penalties for late delivery and non-delivery.

Notations

- i : We use this variable in several ways. For the Channel missions it serves as an index of a *source-destination* pair. For the Contingency and SAAM missions, it serves as an index of the mission i.d. Foe these missions, the source destination pair corresponding to mission 'i' is OD(i).

- j : Aircraft type. It can be one of the following six types : C5, C17, KC10, wide-body commercial (WBC), narrow-body commercial (NBC) and WBP. These differ in their infrastructure requirements, capacity, operational cost etc. We distinguish between the types of aircrafts; Mil and Civ. J_{Mil} is the set of military aircraft types while J_{Civ} is the set of civilian aircrafts types.

- k : Cargo type. It can be one of the following types: 3 types of loads, passengers and fuel.

- l : Location index. We consider a problem with multiple locations. Set of base locations is denoted by $\mathcal{L}$.

- m : Mission type. There are three mission types: Channel, Contingency and SAAM. We label Channel, Contingency and SAAM with 1, 2 and 3 respectively. Each mission is subject to uncertainty, realized by either number of aircraft needed or cargo amount, type and location. *We assume that the cargo from different missions can not mix with each other.*

- r : Route index. Each route begins and ends at a base location (depot) '$O(r)$' and involves visits to a source and destination location (both inbound and outbound). For example, one route may be of the type (Depot – Origin – Dest – Origin – Depot) or (Depot – Origin – Dest – Depot). In some cases the Depot and the origin might also be the same.

- t : Time period. The planning period is divided into smaller chunks of time each of which is then solved independently.

- $y_{j,l,m,t}$: Number of *stage 1* air-crafts of type 'j' allocated to (base) location '$l \in \mathcal{L}$' for mission type 'm' on day t. This is one of the outputs of the program. Note that, stage 1 aircraft are military aircraft as well as civilian aircraft on long term lease. We use $\mathbf{y}$ to denote the allocation vector.

- $u^1_{i,k,m,t}, u^2_{i,k,m,t}$: Unmet demand of cargo type 'k' of OD pair 'i' for mission 'm' in period 't'. The penalty for unmet demand grows linearly as long as the cargo has not been delivered for $t_{i,m}$ days. But beyond that, the penalty is increased. To model this, we divide the unmet demand into two parts. $u^1_{i,k,m,t}$ is the unmet demand that is less than $t_{i,m}$ days old and $u^2_{i,k,m,t}$ is the unmet demand that is more than $t_{i,m}$ days old. To model this, we define a threshold H as follows:

$$H^m_{i,k,t} = \sum_{t-t_{i,m}}^{t} D^m_{i,k,t}(w) \tag{6.1}$$

where $D_{\cdot}$ is the demand (defined later). In general, $t_{i,m}$ may be different for different missions. For example, each contingency may have its own deadline and pricing structure. As explained above, the index i has been overloaded to denote OD pair for channel missions and mission i.d. for SAAM and Contingency missions. In the case of contingency missions, $rdd(i)$ (required delivery date) is used to denote the first day from which the penalty is imposed.

- $x^m_{r,j,t}$: Number of times an aircraft of type 'j' is flown on route 'r' starting on time period 't' for mission 'm'. This is an integer variable but in this implementation we relax it to a continuous variable.

- $\hat{y}^1_{j,l,m,t}$ ($\hat{y}^2_{j,l,m,t}$): Number of stage 1 (stage 2) rented air-crafts of lower cost (higher cost) of type 'j' allocated at (base) location 'l' (for mission type 'm' at time 't'). This is also one of the outputs of the program. Note that, we may choose to have only one type or add even more types of such aircraft by increasing the domain of the superscript.

- $z^m_{i,j,k,r,t}$: For the channel missions, it denotes the amount of cargo of type 'k' for OD pair 'i' transported on aircraft type 'j' using route 'r' in period 't'. The interpretation is modified appropriately for SAAM and Contingency missions. As explained above, the index i has been overloaded to denote OD pair for channel missions and mission i.d. for SAAM and Contingency missions.

  item $A_{j,t}$: Number of hours, an aircraft of type 'j' is available for flying in period 't'. Typically, this will not depend on the period t.

- $C_{r,i,j}$: Capacity of aircraft of type 'j' when flying route 'r' to serve the source-destination pair 'i'. These may depend on the distance of each leg, fuel requirements and other constraints. Note that the payload(capacity) depends on how much fuel one is carrying. Hence, the capacity should be dependent on the position of the OD pair 'i' within the route 'r'.

- $C^k_{r,i,j}$: Capacity of aircraft of type 'j' for carrying cargo of type k when flying route 'r' to serve the source-destination pair 'i'. This is used to handle bulk cargo, oversize cargo in Contingency missions and, space constraints for passengers, etc.

- $\hat{C}_{i,j,t}(\omega)$: Surplus capacity on a aircraft of type j from the SAAM mission to carry load from the source-destination pair i for a channel mission in time period t.

- $D^m_{i,k,t}(\omega)$: Total demand of cargo type 'k' for source-destination pair 'i' for mission 'm' in period 't'. The demand is modeled as a random variable. The units for the demand depend on the mission type. It may be cargo tons, or number of passengers, or number of aircraft as appropriate. In the SAAM missions, the demand is in the number of aircrafts.

- $E_{i,j,l,t}$: Operational Expenses in flying an aircraft of type 'j' on route 'r'.

- $G_1(G_2)$: This is the set of all aircrafts that are available for long term (short term).

- K_j : Index set of all the cargo types that are aggregated for the purposes of imposing the capacity constraint on an aircraft of type 'j'.

- $P^1_{k,m}$ ($P^2_{k,m}$): Penalty per unit weight for late (very late) delivery of a cargo of type 'k' for mission 'm'.

- R^1_j(R^2_j) : Rent of an aircraft of lower cost (higher cost) of type 'j' for one period.

- $S_{i,j}$: For Channel missions, this denotes the index set of direct routes which serve an OD pair 'i' with an aircraft of type j. For Contingency and SAAM missions, this denotes the index set of routes which serve mission 'i' with an aircraft of type j. Note that the military aircrafts ($\{Q(j) = Mil\}$) , must always complete a round-trip; i.e., for them, the route is always (Depot – Origin – Dest – Origin – Depot). The civilian aircrafts ($\{Q(j) = Mil\}$) may or may not fly a round-trip.

- $S1_i$: For Channel missions, this denotes the index set of routes which constitute the first leg of transshipment for OD pair 'i'. Note that this would be flown only by a civilian aircraft. For Contingency and SAAM missions, this denotes the index set of routes which serve mission 'i'.

- $S2_i$: For Channel missions, this denotes the index set of routes which constitute the second leg of transshipment for OD pair 'i'. Note that this must be a military aircraft. For Contingency and SAAM missions, this denotes the index set of routes which serve mission 'i'.

- $\mathcal{TS}$: The set of all OD pairs that require transshipment if civilian aircraft are used.

- $T^m_{r,j,t,t'}(\omega)$: Amount of time the aircraft is occupied in period 't' to complete route 'r' with aircraft 'j' when the route starts in period 't'' while flying mission 'm'. The index ω denotes its stochastic nature.

- $T'^m_{r,j}(\omega)$: Flying time of the aircraft of type 'j' to complete route 'r' while flying mission 'm'. This time is variable because it is affected by the weather condition etc. Hence we use the index ω to denote the stochastic nature.

- $H^1_{i,k,t}$: Maximum unmet demand for source-destination pair i, cargo type k at time t. Unmet demand greater than H1 gets penalized at rate P2.

- μ_j: Permissible utilization of aircraft of type j.
- $\Delta_r^{i,j}$: time to reach the origin of OD pair 'i' on route 'r' using aircraft type 'j'.
- Δ_r^j : time to complete a route 'r' using aircraft type 'j'.
- TP: number of time periods in the problem.
- $Y_{j,l}$: Number of aircrafts of type 'j' at location 'l'.
- $\tilde{Y}_{j,l}^1$: Number of "lease" aircrafts available at location 'l'. Note that only civilian aircrafts can be procured for a long term.
- $\tilde{y}_{j,l}^1$: Number of "lease" aircrafts assigned to location 'l'.

A.1 Stage One Formulation

Stage 1 is a mixed-integer program. The whole problem is sub-divided into smaller chunks of time, say a day. We assume that there is no cost associated with changing the allocations each month. Thus, we propose to solve each problem independently from the previous month (planning period).

Stage 1 determines the values of $y_{j,l,m,t}$, the allocation of aircraft by type, location, mission, and time. The problem is decomposed to facilitate parallel processing of the stochastic realizations, necessitating stage 2 cuts in the formulation. Stage 2 cuts are added successively until a convergence tolerance is met.

$$\begin{aligned}
\min \quad & \sum_j TP(R_j^1 \sum_l \tilde{y}_{j,l}) + \sum_s \pi_s \Theta_s(\mathbf{y}) \\
\text{such that} \quad & \\
\textit{Feasible Allocation1}: \quad & \sum_m y_{j,l,m,t} \leq Y_{j,l} \quad \forall j \in J_{Mil}, l, t \\
\textit{Feasible Allocation2}: \quad & \sum_m y_{j,l,m,t} - \tilde{y}_{j,l} \leq Y_{j,l} \quad \forall j \in J_{Civ}, l, t \\
\textit{Leased , Aircrafts}: \quad & \tilde{y}_{j,l} \leq \tilde{Y_{j,l}} \quad \forall j \in J_{Civ}, l
\end{aligned}$$

$$\textit{Stage 2 Cuts}: \quad \theta_s \geq Opt_s(\mathbf{y}^*) + \sum_{j \in G_1, l, m, t} (A_{j,t}(\omega) v^s_{7,j,l,m,t} + \mu_j v^s_{9,j,l,m})(y_{j,l,m,t} - y^*_{j,l,m,t})$$

where

- $\Theta(\mathbf{y})$: is the expected 'Stage Two' cost of $\mathbf{y}$
- $v^s_{\cdot}$: Optimal dual variables for scenario 's' obtained by solving the 'Stage Two' model.
- π_s: Probability of scenario s.

The objective function seeks to minimize the cost of civilian aircraft allocation, plus the probability-weighted sum of stage 2 cuts. Military aircraft are excluded from the first term because they do not incur allocation costs. The feasible allocation constraints limit the total allocated aircraft to the number available at each base. The stage 2 cuts represent the dual costs from the mission and flight time constraints as affected by the stage 1 y variables. The values of v are fixed in the stage 1 problem.

A.2 Stage Two Formulation

The second stage models the execution of channel, contingency, and SAAM missions for a large number of stochastic realizations. These are approximated by linear programs. The values of $y_{j,l,m,t}$ generated from stage 1 are used as inputs to this program.

$$Opt_s(\mathbf{y},\mathbf{I}) = \min \sum_{j,l,m,t} R_j^2 \hat{y}^2_{j,l,m,t} + \sum_{i,k,t} P^1_{k,1} u^1_{i,k,1,t} + \sum_{i,k,t=rdd(i)}^{rdd(i)+t_{i,2}} P^1_{k,2} u^1_{i,k,2,t}$$
$$+ \sum_{i,k,m,t} P^2_{k,m} u^2_{i,k,m,t} + \sum_{r,j,m,t} E_{r,j} x^m_{r,j,t} + \dots$$

s.t.

Channel

Demand1 :
$$-(u^1_{i,k,1,t-1} + u^2_{i,k,1,t-1}) + \sum_{r \in S_{i,j}} \sum_{j} z^1_{i,j,k,r,t}$$
$$+u^1_{i,k,1,t} + u^2_{i,k,1,t} = D^1_{i,k,t}(\omega) \quad \forall i \notin \mathcal{TS}, k, t$$

Demand2 :
$$-(u^1_{i,k,1,t-1} + u^2_{i,k,1,t-1}) + \sum_{r \in S_{i,j}} \sum_{j \in J_{Mil}} z^1_{i,j,k,r,t}$$
$$+ \sum_{r \in S1_i} \sum_{j \in J_{Civ}} z^1_{i,j,k,r,t} + u^1_{i,k,1,t} + u^2_{i,k,1,t}$$
$$= D^1_{i,k,t}(\omega) \quad \forall i \in \mathcal{TS}, k, t$$

Transshipment :
$$\sum_{r \in S1_i, j \in J_{Civ}} z^1_{i,j,k,r,(t-\Delta_r^{i,j})} - \sum_{r \in S2_i, j \in J_{Mil}} z^1_{i,j,k,r,t}$$
$$= 0 \quad \forall i \in \mathcal{TS}, k, t$$

Aggregate capacity :
$$\sum_{k \in K_j} z^1_{i,j,k,r,(t+\Delta_r^{i,j})} - C^1_{r,i,j} x^1_{r,j,(t-\Delta_r^{i,j})}$$
$$-\hat{C}_{i,j,t}(\omega) x^3_{r,j,(t-\Delta_r^{i,j})} \le 0 \quad \forall i, j, r, t$$

Specific capacity :
$$z^1_{i,j,k,r,(t+\Delta_r^{i,j})} - C^k_{r,i,j} x^1_{r,j,t} \le 0 \quad \forall i, j, k, r, t$$

Price Break :
$$0 \le u^1_{i,k,1,t-1} \le H^1_{i,k,t} \quad \forall i, k, m, t$$

Contingency

Demand1 :
$$-(u^1_{i,k,2,t-1} + u^2_{i,k,2,t-1}) + \sum_{r \in S_{i,j}} \sum_{j} z^2_{i,j,k,r,t}$$
$$+u^1_{i,k,2,t} + u^2_{i,k,2,t} = D^2_{i,k,t}(\omega) \quad \forall i \notin \mathcal{TS}, k, t$$

Demand2 :
$$-(u^1_{i,k,2,t-1} + u^2_{i,k,2,t-1}) + \sum_{r \in S_{i,j}} \sum_{j \in J_{Mil}} z^2_{i,j,k,r,t}$$
$$+ \sum_{r \in S1_i} \sum_{j \in J_{Civ}} z^2_{i,j,k,r,t} + u^1_{i,k,2,t} + u^2_{i,k,2,t}$$
$$= D^2_{i,k,t}(\omega) \forall i \in \mathcal{TS}, k, t$$

Transshipment :
$$\sum_{r \in S1_i, j \in J_{Civ}} z^2_{i,j,k,r,(t-\Delta^{i,j}_r)} - \sum_{r \in S2_i, j \in J_{Mil}} z^2_{i,j,k,r,t}$$
$$= 0 \quad \forall i \in \mathcal{TS}, k, t$$

Aggregate capacity :
$$\sum_{k \in K_j} \sum_{i:OD(i)=odi} z^2_{i,j,k,r,(t+\Delta^{i,j}_r)} - C^2_{r,odi,j} x^2_{r,j,t} \leq 0 \quad \forall odi, j, r, t$$

Specific capacity :
$$\sum_{i:OD(i)=odi} z^2_{i,j,k,r,(t+\Delta^{i,j}_r)} - C^k_{r,i,j} x^2_{r,j,t} \leq 0 \quad \forall odi, j, r, t$$

Price Break :
$$0 \leq u^1_{i,k,2,t-1} \leq H^2_{i,k,t} \quad \forall i, k, m, t$$

SAAM

Demand :
$$\sum_{r \in S_{i,j}} x^3_{r,j,(t-\Delta^{n,j}_r)} = D^3_{i,j,t}(\omega) \quad \forall i, j, t$$

The stage 2 objective function minimizes the sum of short-term rental costs, the cost of late channel and contingency cargo, the cost of very late or undelivered cargo, and the cost of aircraft operations. The demand constraints are represented as cargo inventories for each requirement across time periods; the difference between previous and current inventories, adjusted for deliveries, equals demand. There are separate constraints for cargo that must be directly delivered, and cargo that may be either directly delivered or transshipped. The transshipment constraint ensures cargo delivered in the first leg of a transshipment equals cargo delivered in the second leg. The aggregate capacity constraints limit cargo deliveries by the cumulative capacity of the aircraft assigned for delivery. The specific capacity constraints are similar, but constrain the individual cargo types separately to account for their unique loading characteristics. The price break constraint enforces limits on late cargo.

Mission time constraints ensure that no more aircraft are away from home base than have been allocated. Similarly, the flying time constraint limits aircraft flight hours throughout the model time horizon to their historical maximum.

Appendix B: Effect of Memory Bandwidth on LP Solve Times

We used the Gurobi Optimizer [18] library to do the linear program optimizations. Size of the linear programs in the airfleet assignment problem is much larger than what can fit inside the working set of the process. Linear program optimization will therefore have significant number of memory accesses. Using all the available cores on the chip can cause contention for memory bandwidth and thus cause increase the LP solve times. To quantify the effect of memory bandwidth on LP solve times(stage 2 problem for the 10t Model), we did a number of runs using varying number of cores per node. Figure B.1 shows that the average stage 2 solve time increases as we increase the number of cores per node doing linear program optimizations in parallel. The node used for these runs has Intel 64(Clovertown) 2.33 GHz dual socket quad core processor with 1333MHz front size bus (per socket), 2x4MB L2 cache and 2 GB/core memory. Based on this experiment, we can make two important inferences on doing large size optimizations parallely on a shared memory machine:

- Machines with more memory channels per socket will perform faster solves than the ones with lesser memory channels per socket.
- Not all the cores on the node should be used for LP solves. Some of the nodes can be either left idle or used for other non-memory intensive tasks e.g. communication with external nodes.

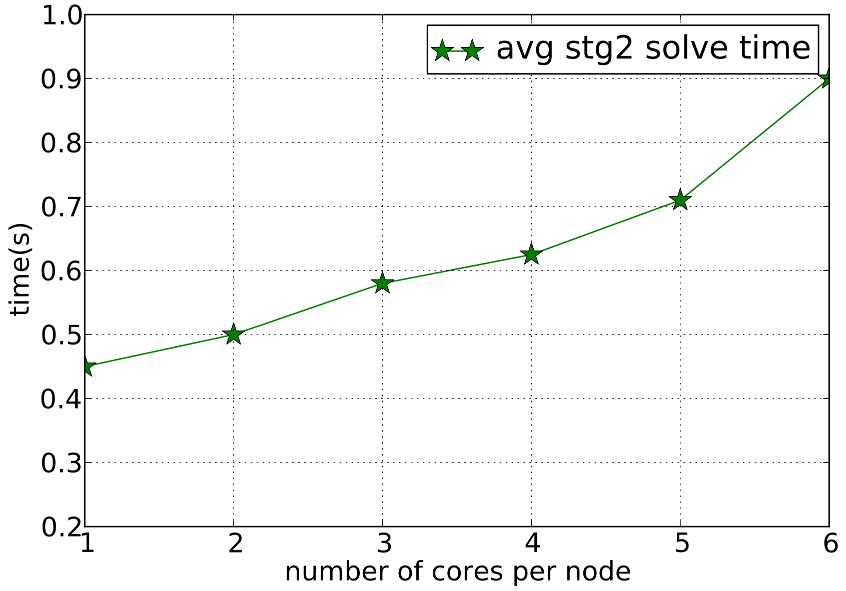


Figure B.1: The impact of artificially constraining memory bandwidth available for an LP solve (10 time period model) on a system with Intel 64(Clovertown) 2.33 GHz dual socket quad core processor with 1333MHz front size bus (per socket), 2x4MB L2 cache and 2 GB/core memory.