FATCOP 2.0: Advanced Features in an Opportunistic Mixed Integer Programming Solver *

Qun Chen
Department of Industrial Engineering
University of Wisconsin, Madison, WI 53706
email: chenq@cae.wisc.edu

Michael C. Ferris
Computer Sciences Department
University of Wisconsin, Madison, WI 53706
email: ferris@cs.wisc.edu

Jeff T. Linderoth
Mathematics and Computer Science Division
Argonne National Laboratory, Argonne, IL 60439
email: linderot@mcs.anl.gov

December 20, 1999

Abstract

We describe FATCOP 2.0, a new parallel mixed integer program solver that works in an opportunistic computing environment provided by the Condor resource management system. We outline changes to the search strategy of FATCOP 1.0 that are necessary to improve resource utilization, together with new techniques to exploit heterogeneous resources. We detail several advanced features in the code that are necessary for successful solution of a variety of mixed integer test problems, along with the different usage schemes that are pertinent to our particular computing environment. Computational results demonstrating the effects of the changes are provided and used to generate effective default strategies for the FATCOP solver.

*This material is based on research supported in part by National Science Foundation Grants CDA-9726385 and CCR-9972372, the Air Force Office of Scientific Research Grant F49620-98-1-0417 and Microsoft Corporation.

1 Introduction

Many practical optimization problems involve a mixture of continuous and discrete variables. Examples of such problems abound in applications; for example scheduling and location problems, covering and partitioning problems and allocation models. While many of these problems also contain nonlinear relationships amongst the variables, a large number of interesting examples can be effectively modeled using linear relationships along with integer variables. Such problems are typically called mixed integer programs [13].

Typically, these problems are solved using a branch-and-bound approach [7]. First of all, the integer constraints are relaxed to simple bound constraints, resulting in a linear programming relaxation. This relaxation is solved, typically by a version of the simplex method, and the solution is tested to determine if the integrality constraints are satisfied. If not, one of the variables that violates integrality is chosen, two subproblems are generated each with an extra constraint that precludes the current infeasible solution. However, any integer solution will be feasible for one of the subproblems. Branching in this way, generates a huge number of linear programs that need to be processed in order to solve the original problem. Subproblems can be discarded (fathomed) if

1. the subproblem has an integer solution,

2. the subproblem is infeasible, or

3. the linear subproblem has worse objective value than a currently available integer solution (bounding).

In many realistic problems, the size of the branch-and-bound tree is enormous, and requires huge amounts of computing resources to solve the underlying program. Furthermore, due to bounding and fathoming, the shape of the tree is generally very irregular and cannot be determined a-priori. As such, efficient use of large numbers of processors in this context is difficult.

FATCOP 1.0 [2] is an implementation of a branch-and-bound algorithm that attempts to be both efficient and able to solve difficult MIP's. A key feature of FATCOP 1.0 is that it runs in an opportunistic environment where the computational resources are provided by Condor [11].

Condor is a resource manager that locates idle machines on a local or wide area network and delivers them to an application (such as FATCOP) as a computational resource. Originally, these resources were provided for batch processing of long-running, computationally intensive programs. Each machine in a Condor pool has an owner that specifies the conditions under which a machine is made available. For example, most owners require a job to vacate a machine when the owner returns to use the machine. It is important to note that a resource user does not need to have an account on the machines on which the job executes. Furthermore, the footprint of the job is small since all of the data is stored on the submitting machine and accessed via remote procedure calls. Thus, only the processing power of the "Condor" machine is utilized. More recently, an extension of Condor [14] has allowed a collection of Condor resources to be treated as a single computational entity and programmatically controlled using primitives from PVM [6].

FATCOP (FAult Tolerant COndor Pvm) 1.0 [2] was developed with the aim of exploiting the opportunistic resources provided by Condor in a fault tolerant fashion. To provide this fault tolerance, FATCOP is written in the master-worker paradigm. The master-worker framework has three abstractions, namely that of a master, a worker, and a task. In FATCOP 1.0, a worker is a machine loaded with the root linear programming relaxation, a task consists of solving linear programming relaxations with added bound restrictions, and the master is a controlling process that deals with all messages coming from all workers and all tasks. Essentially, multiple branches of the search tree are explored concurrently, with the master determining from a work pool which node to explore next, noting the best solution found and fathoming parts of the tree that are inconsequential. Linear programming subproblems comprise the tasks that are solved on workers provided by Condor.

A basic premise of FATCOP is to greedily use as many resources as possible to solve the underlying problem. Particular attention is paid at the master to issues relating to disappearance of resources and recovery of work assigned to these resources. For long running problems, facilities exist that allow FATCOP to recover from most failures using a mechanism that periodically saves the branch-and-bound tree to disk (checkpointing).

While FATCOP 1.0 can successfully process many mixed integer programs, several deficiencies in the algorithm and the implementation limit its effectiveness on harder problems, several of which became apparent after extensive testing. These are as follows:

1. Inefficient use of resources - the chunks of work are too small, leading to contention effects at the master

2. Poor mixed integer programming technology

3. Not enough resources used
4. Poor linear programming codes
5. Difficulties associated with code maintenance

This paper describes a series of ways in which FATCOP 1.0 can be improved, and details the implementation of FATCOP 2.0. A schematic figure showing the basic flow of information and an overview of the implementation of FATCOP 2.0 is given in Figure 1. The figure shows that the flow of communication between the master and a (prototypical) worker occurs only when the worker is initialized. Tasks run on a worker, get their initial data (MIP problem, cuts and pseudocosts) from the worker they are assigned to, and their specific task information (subtree to work on, incumbent solution) directly from the master. New pseudocost and cut information from the task is saved on the current worker and hence may be used by a new task that runs on this worker. The task also sends solution information (and newly generated cuts and pseudocosts) back to the master so that it can update its work pool, global cut and pseudocost pool, and incumbent solution.

We begin in Section 2 with a description of the advanced MIP features that we have added to FATCOP 2.0. These features are applied in nonstandard ways since out search strategy is updated to have much larger grained tasks involving branches of the search tree (as opposed to nodes of the search tree). In Section 3, we outline the two major resource management changes that are critical for solving difficult MIP's. The first change utilizes a computational framework embedded in an API, MW (short for Master-Worker), thereby reducing the complexity of the FATCOP code. It also allows different computational setups to be used, such as providing heterogeneous machines as workers or replacing PVM messages with file transfers. The second change allows a variety of linear programming software to be used interchangeably on the workers, thereby increasing the efficiency of subproblem solution. The benefits of all these changes are demonstrated in Section 4 where a variety of tests are carried out on a representative set of MIP test problems. We finish with some preliminary conclusions of the work in Section 5.

2 Advanced MIP features

While the use of many computational resources can sometimes alleviate the need for good algorithms, the classes of problems we are interested in solving require sophisticated MIP features in addition to computational power.

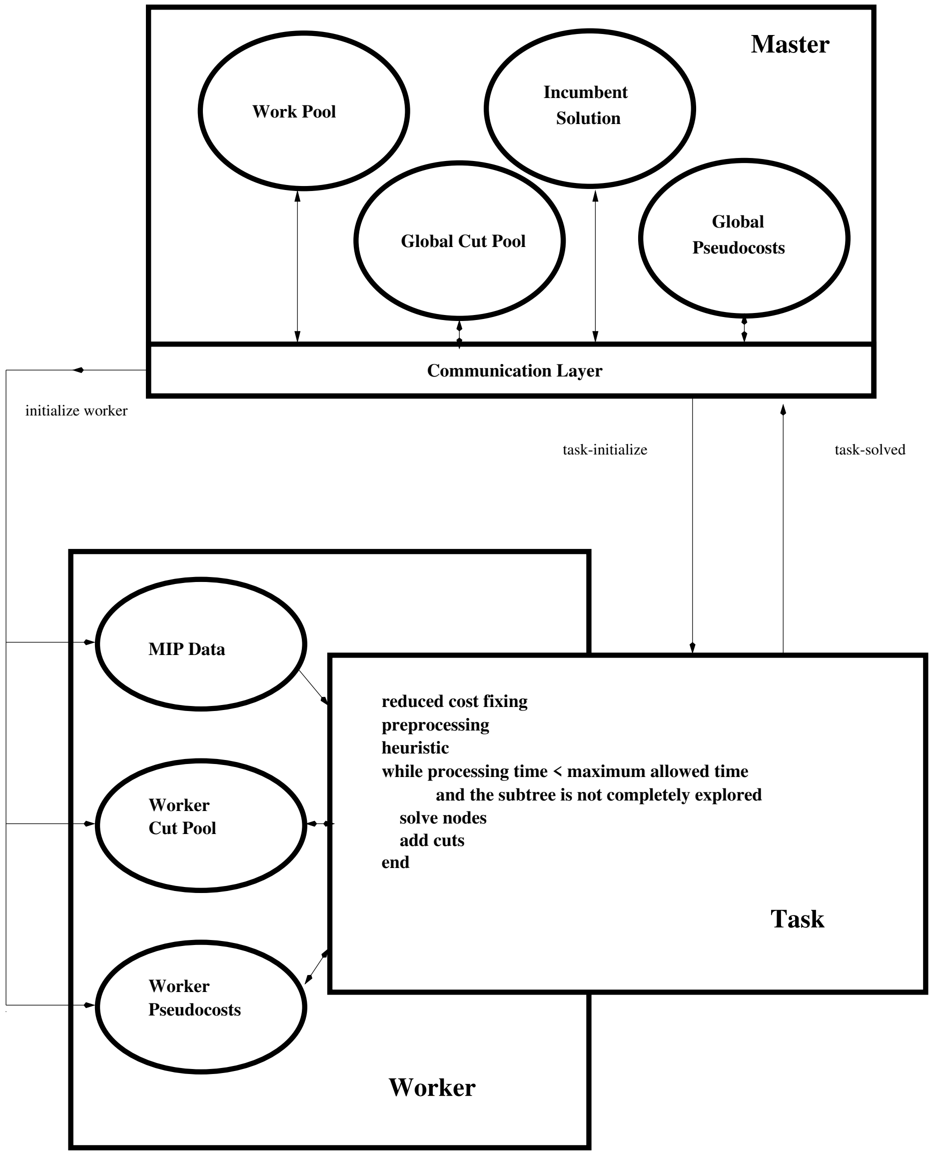

Figure 1: FATCOP overview

Several of these features enable problem solution in fractions of the time needed for a basic branch-and-bound algorithm. The aim of adding these features to FATCOP 2.0 is to ensure that a complex MIP code uses dynamic resources efficiently, and to deal with the computational complexity of sharing local and global information within a continually changing computing framework.

2.1 Search strategies

A critical problem in FATCOP 1.0 arises from contention issues at the master. In FATCOP 1.0, each worker solves two linear programs before reporting back to the master. The master needs to deal with new information coming from the problems the workers have solved, as well as issues related to the addition or deletion of hosts from the virtual machine. When the linear program relaxations are relatively time consuming, this contention is not limiting, but in many cases, the relaxations solve extremely quickly due to advanced basis information.

To alleviate this problem, FATCOP 2.0 has a new notion of a worker and a task. A task is a subtree for the worker to process, along with a limit on the number of linear programs that can be solved, or a limit on the processing time. The data associated with a task includes what subtree is to be processed along with strategy information for processing this tree, such as the value of the best feasible solution and the node and time limits for processing the task. Note that the subtrees passed to each worker are distinct.

However, a worker now has to store all the MIP information itself, as well as the linear program data. Thus, a worker has a state that consists of all this information (that is passed to it when the worker is initialized), along with any extra information (such as cuts or pseudocosts) that is generated by any task that runs on the worker.

The time limit feature generates new issues, namely how to pass back a partially explored subtree to the master. In order to limit the amount of information passed back, we use depth-first-search to explore the subtrees, since then a small stack can be passed back to the master encoding the remaining unexplored parts of subtree. Furthermore, it is also easy to use the small changes to the LP relaxations in such a search mechanism to improve the speed of their solution. Finally, any "local information" that is generated in the subtree is valid and typically is most useful in the subtree at hand. As examples of this last point, we point to the reduced cost fixing and preprocessing techniques that we outline later in this section.

Most of the issues about search strategies of the master program were dealt with in [2]. Two changes are of note. The first is that instead of generating the first N nodes in the master, now only the first linear programming relaxation is solved at the master. The optimal basis from this relaxation is sent to all workers, so that they may solve all subsequent nodes efficiently. Furthermore, whenever there are less than a certain number of nodes in the work pool, we switch from a time limit in the worker to an LP solve limit of 1. This allows the work pool to grow rapidly in size.

A benefit of FATCOP 2.0 is that the amount of information passed back from a worker is now much smaller. This makes it possible to have much larger work pools, and allows FATCOP 2.0 to use best bound as its default node selection strategy. However when the size of the work pool reaches an upper limit, we switch the node selection strategy to use "deepest-node" in tree, with the expectation that the subtrees rooted at these nodes are likely to be completely explored by the worker, thus leading to a decrease in the size of the work pool.

2.2 Cutting planes

If the solution to the linear programming relaxation does not satisfy the integrality requirements, instead of generating new subproblems (branching), one may attempt to find an inequality that "cuts off" the relaxed solution. That is, an inequality that is not valid for the relaxed solution, but is valid for all integer solutions. Such an inequality is called a *cutting plane*. Adding cutting planes to the relaxation can result in an improved lower bound for the relaxation, which in turn may mean that the linear subproblem can be fathomed without having to resort to branching.

There are many different classes of cutting planes. FATCOP 2.0 includes two classes – *knapsack cover inequalities* [3] and *flow cover inequalities* [15]. Knapsack covers and flow covers inequalities are derived from structures that are present in many, but not all, MIP instances. This implies that for some instances, FATCOP 2.0 will be able to generate useful cutting planes, and for other instances it will not.

The problem of finding a valid inequality of a particular class that cuts off the relaxed solution is known as the *separation problem*. For both classes of inequalities used in FATCOP 2.0, the separation problem is NP-Complete, so a heuristic procedure is used for finding violated inequalities.

Cutting planes represent a new challenge for FATCOP 2.0. They provide globally valid information about the problem that is locally generated. Namely, a cutting plane generated at one processor may be used to exclude relaxed solutions occurring at another processor. The question arises of how to distribute the cutting plane information.

We have chosen to attach this information to the worker by creating a cut pool on the worker. All newly generated cuts get sent to the master when a task completes, but this information is only sent to new workers, not to existing workers. Thus each worker carries cut information that was generated by the tasks that have run on the worker, but never receives new cuts from the master.

2.3 Pseudocosts

If inequalities that cut off the relaxed solution cannot be found, then branching must be performed. In FATCOP, branching is performed by selecting a variable j in the relaxed solution that does not satisfy the integrality requirements and creating two subproblems. In one subproblem variable j is required to be less than its value in the relaxed solution, and in the other subproblem, variable j is required to be more than its value in the relaxed solution

In a relaxed solution, there may be many variables that do not satisfy integrality requirements. The goal of branching is to choose an variable that will most improve the subproblems' objective function. The rationale behind this goal is that if a subproblem's objective function increases by a large amount, then it may be possible to fathom the subproblem. Pseudocosts are information that aid in choosing among the many fractional variables.

Pseudocosts pose a challenge to FATCOP 2.0 in exactly the same way as cutting planes, in that they are globally useful information that is generated locally. As such, we choose to distribute pseudocosts in a manner similar to that for cutting planes. All new pseudocosts get sent to the master when a task completes, but this information is only sent to new workers, not to existing workers.

2.4 Heuristics

A heuristic is a procedure that attempts to generate a feasible integral solution. Feasible solutions are important not only for their own sake, but also as they provide an upper bound on the optimal solution of the problem. With this upper bound, subproblems may be fathomed, and techniques such as reduced cost fixing can be performed.

There are very few general purposes heuristics for mixed integer programs. One simple, yet effective heuristic is known as the *diving* heuristic.

In the diving heuristic, some integer variables are fixed and the linear program resolved. The fixing and resolving is iterated until either an integral solution is found or the linear program becomes infeasible.

We have included a diving heuristic in FATCOP 2.0. The diving heuristic can be quite time consuming – too time consuming to be performed at every node of the branch and bound tree. In FATCOP 2.0, since a task is to explore an entire subtree for a specified time limit, this also gives a convenient way to decide from which nodes to perform the diving heuristic. Namely, the diving heuristic is performed starting from the root node of each task.

Preliminary testing revealed that for some instances this strategy for deciding when to perform the heuristic was also too time consuming. Therefore, if the total time spent in carrying out the diving heuristic grows larger than 20% of the total computation time, the diving heuristic is deactivated. Once the time drops to below 10%, the diving heuristic is reactivated.

2.5 Preprocessing

Preprocessing refers to a set of reformulations performed on a problem instance to enhance the solution process. It has been shown to be a very effective way of improving integer programming formulations prior to and during branch-and-bound [16]. FATCOP 1.0 preprocesses the root problem by identifying infeasibility and redundancies, tightening bounds on variables, and improving coefficients of constraints [2]. In FATCOP 2.0, we extend this procedure to apply preprocessing at the root node of every task sent to a worker. This is due to the change in search strategy, whereby each task consists of exploring a subtree of the search tree rooted at the passed node.

At a node in the branch-and-bound tree where the optimal solution of its LP relaxation is fractional, we first apply a standard reduced cost fixing procedure [5]. This procedure fixes integer variables to their upper or lower bounds by comparing their reduced costs to the gap between a linear programming solution value and the current problem best upper bound. After this, we perform preprocessing on the new problem. Finally, the diving heuristic described above is applied to find a feasible integer solution. Note that we can reverse the order of reduced cost fixing and node preprocessing in the hope that reduced cost fixing may work better on a preprocessed model. However, the current implementation chooses instead to take advantage of preprocessing a model that possibly has more variables fixed by the reduced cost fixing procedure.

In a sequential branch-and-bound MIP program, node preprocessing is usually considered too expensive. However, in FATCOP 2.0, every worker explores a subtree of problems. The cost of preprocessing is amortized over the subsequent LP solves. Preprocessing may improve the lower bound of this subtree, and increase the chance of pruning the subtree locally; however, the effects of node preprocessing are problem dependent. Therefore, we leave node preprocessing as an option in FATCOP 2.0.

The key issue is that the search strategy in FATCOP 2.0 generates a piece of work whose granularity is sufficiently large for extensive problem reformulations to be effective and not too costly in the overall solution process. All the approaches outlined above are implemented to exploit the locality of the subproblems that are solved as part of a task, and in our implementation are carried out at many more nodes of the search tree than is usual in a sequential code. The benefits and drawbacks of this choice are further explored in Section 4.

3 Resource management improvements

3.1 MW framework

FATCOP 1.0 is implemented using Condor-PVM, an extension of the PVM programming environment that allows resources provided by Condor to be treated as a single (parallel) machine. As outlined in the introduction, FATCOP utilizes the master-worker computing paradigm. Thus many of the details relating to acquiring and relinquishing resources, as well as communicating with workers are dealt with explicitly using specific PVM and Condor primitives. Many of the features, and several extensions, of the resource management and communication procedures in FATCOP 1.0 have been incorporated into a new software API, MW [8], that can be used for any master-worker algorithm. Since this abstraction shields all the platform specific details from an application code, FATCOP 2.0 was redesigned to use this API, resulting in a much simpler, easier to maintain, code.

Other benefits also accrue that are pertinent to this work as well. First, MW provides the application (in this case FATCOP) with details of resource utilization that can be analyzed to improve efficiency (see Section 4). Secondly, new features of MW immediately become available for use in FATCOP. As an example, a new instantiation of MW that is built upon a communication model that uses disk files (instead of PVM messages) can now be used by FATCOP without any change to the FATCOP source code. Since this instantiation also uses standard Condor jobs instead of PVM tasks for the workers, facilities such as worker checkpointing that are unavailable in the PVM environment also become usable in the file environment. (Condor provides a checkpointing mechanism whereby jobs are frozen, vacated from the machine, and migrated to another idle machine and restarted.)

Also, other potential instantiations of MW utilizing MPI or NEXUS for communication or Globus for resource management are immediately available to FATCOP. Thirdly, FATCOP can also drive new developments to MW, such as the requirement for a broadcast mechanism in MW to allow dispersion of new cuts to all workers. Such extension would undoubtedly benefit other MW applications such as those outlined in [8].

3.2 Heterogeneity

The MW framework is built upon the model of requesting more resources as soon as resources are delivered. In order to increase the amount of resources available to the code, we exploited the ability of MW to run in a heterogeneous environment. In this way, the code garnered computational resources from a variety of machines including Sun SPARC machines running Solaris, INTEL machines running Solaris, and INTEL machines running Linux. While INTEL machines running NT are in the Condor pool, currently the MW framework is unavailable on this platform. To effect usage of workers on different architectures, all we needed to do was:

1. Compile each worker program for the specific architectures that it will run on.

2. Generate a new "job description file" for FATCOP 2.0 that details the computational resources that are feasible to use.

Since the source code for the solver SOPLEX [17] is available, compiling the worker code on several platforms is straightforward. The benefits of this increase in number of workers is shown in Section 4.

It became clear that while SOPLEX is an effective linear programming code, commercial codes such as CPLEX [10], OSL [9] and XPRESS [4] significantly outperform SOPLEX for solving the LP problem relaxations. In many cases, several copies of these solvers are available to a user of FATCOP 2.0 and so we updated the design of the code to allow a variety of LP solvers to be used interchangeably. Thus, at any given time, several of the workers may be using CPLEX, while others are using XPRESS and others still are using SOPLEX. The LP interface deals carefully with issues such as how many copies of CPLEX are allowed to run concurrently (for example if a network license is available), what machines are licensed for XPRESS, and what architectures can OSL be ran upon. If none of the faster solvers are available, SOPLEX is used as the default solver.

4 Results

In this section, results on the performance of FATCOP 2.0 are reported. Due to the asynchronous nature of the algorithm, the nondeterminism of running times of various components of the algorithm, and the nondeterminism of the communication times between processors, the order in which the nodes are searched and the number of nodes searched can vary significantly when solving the same problem instance. Other researchers have noticed the stochastic behavior of asynchronous parallel branch and bound implementations [5]. Running asynchronous algorithms in the dynamic, heterogeneous, environment provided by Condor only increases this variance. As such, for all the computational experiments, each instance was run a number of times in an effort to reduce this variance so that meaningful conclusions can be drawn from the results.

The results of FATCOP 2.0 are given on a number of test instances taken from the MIPLIB set [1]. A time limit of 120 seconds was set on each task. If the number of unevaluated tasks at the master fell below two times the number of workers, a limit of one linear program solution was enforced for the task. Unless explicitly stated otherwise, all the advanced features of FATCOP 2.0 described in Section 2 were employed.

4.1 Assessing node preprocessing

It is well known that lifted knapsack covers, flow covers and diving heuristics are effective in solving MIP problems [3, 15, 12]. However, the reported overall benefits of node preprocessing are less clear due to the amount of computing time they may take. A key issue is that node preprocessing is too expensive to carry out at every node. Since our tasks now correspond to subtrees of the brand-and-bound tree, it makes sense in this setting to experiment with preprocessing just at the root nodes of these subtrees. In this section we report results for experiments that ran a number of MIP problems with node preprocessing turned off and on, while all other advanced features (cutting planes, diving heuristics, reduced cost fixing and root preprocessing) were turned on.

The algorithmic parameters that were used are as stated above. Each instance was replicated three times. We report the number of nodes, the wall clock time and the average number of processors used with and without node preprocessing in Table 1. The average number of processors was computed

Name	Node preprocessing			No node preprocessing		
	Nodes	Time	$\bar{P}$	Nodes	Time	$\bar{P}$
egout	11	11.3	2	26	12.3	2
l152lav	3530	345.6	4	5801	468.3	7
p0548	215	16.1	2	222	20.0	2
p2756	3781	2907.8	61	3963	3614.2	39
cap6000	119232	6530.2	30	129720	3317.0	30

Table 1: Effects of Node Preprocessing

as

$$\bar{P} = \frac{\sum_{k=1}^{P_{max}} k\tau_k}{T}, \tag{1}$$

where τ_k is the total time when the FATCOP job has k workers, T is the total execution time for the job, P_{max} is the number of available machines in the Condor's pool.

As expected, all the test problems were solved in less nodes with node preprocessing, since the subtrees were pruned more effectively in the branch-and-bound process. An interesting observation is that it took longer to solve cap6000 even though the search tree is smaller with node preprocessing. In fact, node preprocessing combined with local reduced cost fixing worked very effectively on this problem. After the first integer feasible solution was found, preprocessing and reduced cost fixing usually can fix more than half of the binary variables at the root node of a subtree. But the problem is that cap6000 has a very large LP relaxation. The cost to reload the preprocessed LP model into the LP solver is significant compared with task grain size. Similarly, for problem p2756, although the execution time is shorter with node preprocessing, the runs with node preprocessing got many more machines than the runs without node preprocessing. These observations suggests a better implementation for modifying a formulation in a LP solver is necessary.

4.2 Grain size and master contention

A potential drawback of a master-worker program is the master *bottleneck* problem. When using a large number of processors, the master can become a bottleneck in processing the returned information, thus keeping workers idle for large amounts of time. In FATCOP 2.0, we deal with this problem

grain size	$\bar{E}$	Nodes	Time	$\bar{P}$
2	0.16	809945	1335.8	45
100	0.64	1479350	743.9	25
200	0.61	1938241	1053.2	29

Table 2: Results for vpm2 by varying worker grain size

Machine architecture	Nodes	Time	$\bar{P}$
SUN4	16910	763.5	24
X86	20364	1290.5	16
SUN4 and X86	23840	636.7	38

Table 3: Results for 10teams using heterogeneous machines

by allowing each worker solve a subtree in a fixed amount of time. The rule to choose an appropriate grain size at worker is arbitrary. In this section we show the results for FATCOP 2.0 on vpm2 by varying worker grain size.

We ran FATCOP 2.0 on vpm2 with worker grain size 2, 100 and 200 seconds respectively. In each case, we ran three replications employing all advanced features described in Section 2. The results are reported in Table 2. A grain size of two seconds had a very low worker utilization. Since vpm2 has very small LP relaxations, each worker is able to finish its work quickly, resulting in a large amount of result messages queued at the master. The node utilization corresponding to grain size of 100 seconds is satisfactory. Increasing grain size does not improve node utilization further. As stated in [2], all Condor-PVM programs risk losing the results of their work if a worker is suspended or deleted from the virtual machine. Taking this into consideration, we prefer a smaller worker grain size in order not to lose too much computation when a worker disappears from the virtual machine. We believe a grain size about 100 strikes a good balance between contention and loss of computation and is appropriate for the default.

4.3 Heterogeneity

In this section we show how FATCOP 2.0 exploits heterogeneous resources, including both heterogeneous machines and LP solvers. We ran the problem 10teams on a pool of Sun SPARC machines running Solaris (SUN4), a pool of INTEL machines running Solaris (X86), and a pool of both types of machine. Note that the worker executables are different on these different

LP solver	Nodes	Time	$\bar{P}$
SOPLEX only	3623	19125.0	43
SOPLEX and CPLEX	3661	6626.2	16

Table 4: Results for air04 using heterogeneous LP solvers

architectures. Each instance was replicated three times and we report the results in Table 3. Clearly, FATCOP was able to get more workers when requesting machines from two architecture classes.

We also ran some experiments to show the effects of heterogeneous LP solvers. We solved the problem air04 with SOPLEX only, and both SOPLEX and CPLEX. We limited the maximum number of CPLEX copies to 10 in the latter case. Results are shown in Table 4. The problem air04 has very large LP relaxations, so the worker running SOPLEX usually can only solve one LP in the specified grain size (120 seconds), while a worker running CPLEX is able to evaluate a number of nodes in the depth first fashion outlined previously. We notice from Table 4 that using CPLEX and SOPLEX the problem was solved three times faster using less machines compared with using SOPLEX only.

4.4 Raw performance

Based on the experiments outlined above, we set appropriate choices of the parameters of our algorithm. In this subsection, we attempt to show that FATCOP 2.0 works well on a variety of test problems.

In Table 5, for each test instance, we report the number of nodes, solution time, average worker efficiency $\bar{E}$, and average number of processors $\bar{P}$. The average efficiency was computed as the ratio of the total time workers spent performing tasks to the total time the workers were available to perform tasks. For each of the statistics, we report the minimum, average, and maximum values over all trials.

Figure 2 shows, for one particular trial and instance, the number of participating processors. Figure 3 shows, for the same trial and instance, the instantaneous worker efficiency, measured as $\sum_{k=1}^{n_t} l_k^t / n_t$, where n_t is the number of processors participating at time t and l_k^t is the *load average* of the processor k at time t. The load average, computed using the UNIX command `uptime`, and number of participating processors were sampled at 30 second intervals during the run.

The efficiency of a run may be less than "ideal" (1.0) due to

Instance	Statistic	$\bar{E}$	$\bar{P}$	Nodes	Time
pk1	minimum	63.5	6.7	2786670	1275
	average	72.3	13.4	3511512	2138
	maximum	78.9	19.0	4048204	4353
l152lav	minimum	82.4	1.9	1845	53
	average	91.6	6.5	2824	572
	maximum	98.1	16.9	3919	2080
stein45	minimum	45.6	2.9	181986	499
	average	64.1	12.3	194472	1018
	maximum	88.7	25.0	215029	1694
p2756	minimum	44.3	7.6	6249	866
	average	51.3	21.1	9687	1595
	maximum	61.9	14.4	14115	2202
air05	minimum	18.9	28	7779	3503
	average	43.6	42.4	8180	5771
	maximum	69.3	58.6	8419	9549
bell5	minimum	33.3	6.6	518875	359
	average	42.0	11.8	1600810	1098
	maximum	54.2	14.4	4578776	2464
gesa2_o	minimum	72.7	43.2	6207993	6951
	average	84.9	62.5	8214031	10074
	maximum	94.6	86.3	9693518	13198

Table 5: Performance of FATCOP 2.0

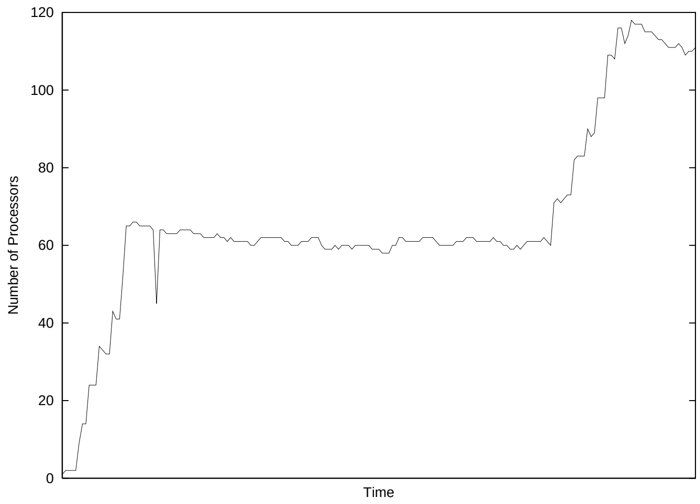

Figure 2: Average Number of Processors Participating in Solving gesa2_o

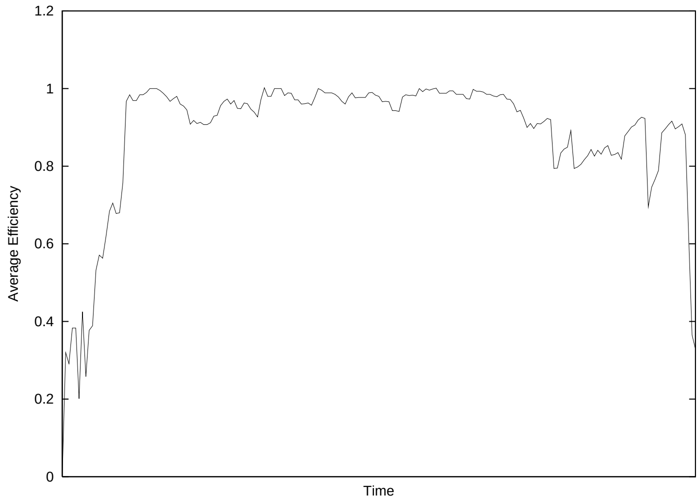

Figure 3: Average Worker Efficiency During Solution of gesa2_o

- Contention – The workers are idle during the time they send the results of their task to the master until they receive the next task. If the master needs to respond to many requests, workers may idle for long periods waiting for new work, thus reducing efficiency.

- Starvation – There are not enough active tasks in the work pool for all the participating workers.

- Inaccuracy of measurements – The load average reported by the UNIX operating system is computed as the average number of processing jobs during the last minute, so even though a processor is working on a task, the reported load average may be less than 1.0.

5 Conclusion

The results reported in this paper show that FATCOP is both an effective MIP solver for a variety of test problems arising in the literature, and an efficient user of opportunistic resources. Further experiments with FATCOP will be made to investigate how well the ideas presented scale with an increased number of available resources. Also, we intend to investigate the use of different cutting planes, as well as further exploitation of the local nature of information when performing a task. Other extension to the code include adding the ability for user defined heuristics and branching rules.

Acknowledgements

The authors would like to thank Martin Savelsbergh for providing code from which the cutting plane code was derived.

References

[1] R. E. Bixby, S. Ceria, C. M. McZeal, and M.W.P. Savelsbergh. MIPLIB 3.0. http://www.caam.rice.edu/~bixby/miplib/miplib.html.

[2] Q. Chen and M. C. Ferris. FATCOP: A fault tolerant Condor-PVM mixed integer program solver. Mathematical Programming Technical Report 99-05, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1999.

[3] H. Crowder, E. L. Johnson, and M. W. Padberg. Solving large scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.

[4] Dash Associates, Blisworth House, Blisworth, Northants, UK. *XPRESS-MP User Guide*. http://www.dashopt.com/.

[5] J. Eckstein. Parallel branch-and-bound algorithms for general mixed integer programming on the CM-5. *SIAM Journal on Optimization*, 4:794–814, 1994.

[6] G. A. Geist, A. L. Beguelin, J. J. Dongarra, W. Jiang, R. Manchek, and V. S. Sunderam. *PVM: Parallel Virtual Machine*. MIT Press, Cambridge, Massachusetts, 1994.

[7] B. Gendron and T. G. Crainic. Parallel branch-and-bound algorithms: Survey and synthesis. *Operations Research*, 42:1042–1060, 1994.

[8] J.-P. Goux, J. Linderoth, and M. Yoder. Metacomputing and the master-worker paradigm. Technical report, Argonne National Laboratory, 1999.

[9] M. S. Hung, W. O. Rom, and A. D. Warren. *Handbook for IBM OSL*. Boyd and Fraser, Danvers, Massachusetts, 1994.

[10] ILOG CPLEX Division, Incline Village, Nevada. *CPLEX Optimizer*. http://www.cplex.com/.

[11] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor: A hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, June 1988.

[12] G. L. Nemhauser, M. W. P. Savelsbergh, and G. S. Sigismondi. MINTO, a mixed integer optimizer. *Operations Research Letters*, 1994.

[13] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, New York, NY, 1988.

[14] J. Pruyne and M. Livny. Interfacing Condor and PVM to harness the cycles of workstation clusters. *Journal on Future Generations of Computer Systems*, 12, 1996.

[15] T. J. Van Roy and L. A. Wolsey. Solving mixed integer 0-1 programs by automatic reformulation. *Operations Research*, 35:45–57, 1987.

[16] M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.

[17] R. Wunderling. SOPLEX: User documentation. http://www.zib.de/Optimization/Software/Soplex/.