FATCOP: A Fault Tolerant Condor-PVM Mixed Integer Program Solver *

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

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

March 17, 1999

Abstract

We describe FATCOP, a new parallel mixed integer program solver written in PVM. The implementation uses the Condor resource management system to provide a virtual machine composed of otherwise idle computers. The new solver differs from previous parallel branch-and-bound work by implementing a general purpose parallel mixed integer programming algorithm in an opportunistic multiple processor environment, as opposed to a conventional dedicated environment. It shows how to make effective use of resources as they become available while ensuring the program tolerates resource retreat. The solver performs well on test problems arising from real applications, and is particularly useful for solving long-running hard mixed integer programming problems.

1 Introduction

Mixed integer programming (MIP) problems are difficult and commonplace. For many of these hard problems, only small instances can be solved in a reasonable amount of time on sequential computers. Therefore, mixed integer programming has been a frequently cited application of parallel computing. Most available general-purpose large-scale MIP codes use branch-and-bound (BB) to search for an optimal integer solution by solving a sequence of related linear programming (LP) relaxations that allow possible fractional values. This paper discusses a new parallel mixed integer program solver, written in PVM, that runs in the opportunistic computing environment provided by the Condor resource management system.

Parallel BB algorithms for MIP have attracted many researchers [9, 13, 15]. Most parallel BB programs were developed to run using large centralized mainframes or big-iron supercomputers that are typically very expensive. Users of these facilities usually only have a certain amount of time allotted to them and have to wait their turn to run their jobs. However, due to the decreasing cost of lower-end workstations, large heterogeneous clusters of workstations connected through fast local networks are becoming common in work places such as universities, research institutions, etc. In this paper we shall refer the former resources as

*This material is based on research supported in part by National Science Foundation Grants CDA-9726385 and CCR-9619765 and Air Force Office of Scientific Research Grant F49620-98-1-0417.

dedicated resources and the later as distributed ownership resources. The principal goal of the research outlined in this paper is to exploit distributed ownership resources to solve large mixed integer programs. We believe that a parallel BB program developed to use this type of resource will become a widely used tool for such problems.

PVM (parallel virtual machine) [12] is a parallel programming environment that allows a heterogeneous network of computers to appear as a single concurrent computational resource. It provides a unified framework within which parallel programs for a heterogeneous collection of machines can be developed in an efficient manner. However PVM is not sufficient for us to develop an efficient parallel BB program in a distributed ownership environment. The machines in such an environment are usually dedicated to the exclusive use of individuals. The application programming interface defined by PVM requires users explicitly select machines on which to run their programs. Therefore they must have permission to access the selected machines and cannot be expected to know the load on the machines in advance. Furthermore, when a machine is claimed by a PVM program, the required resources in the machine will be "occupied" during the life cycle of the program. This is not a desirable situation when the machine is owned by a person different from the user of the program.

Condor [6, 10, 17], a distributed resource management system, can help to overcome the problems a normal PVM program runs into. Condor manages large heterogeneous clusters of machines in an attempt to make use of the idle cycles of some users' machines to help satisfy the needs of others who have computing extensive jobs. It was first developed to run sequential programs. The current version of Condor provides a Condor-PVM framework to run parallel programs written in PVM in a distributed ownership environment. In such programs, Condor is used to dynamically construct PVM virtual machine out of non-dedicated desktop machines on the network. Condor allows users' programs to run on any machine in the pool of machines managed by Condor, regardless of whether the user submitting the job has an account there or not, and guarantees that heavily loaded machines will not be selected for an application. To protect a machine's owner, Condor interrupts a running job in a machine submitted by another user after the machine's owner returns. Since resources managed by Condor are competed for by owners and many other Condor users, we refer to such resources as Condor's *opportunistic resources* and the Condor-PVM parallel programming environment as the Condor-PVM *opportunistic environment*.

FATCOP represents a first attempt to develop a general purpose parallel solver for mixed integer programs in Condor's opportunistic environment. It is implemented on top of SOPLEX , a simplex object-oriented linear programming solver developed by Roland Wunderling [26]. FATCOP is written in the C++ programming language with calls to PVM library. It is designed to make best use of participating resources managed by Condor while handling resource retreat carefully in order to ensure the eventual and correct completion of a FATCOP job. Key features of FATCOP include:

- parallel implementation under Condor-PVM framework;
- greedy utilization of Condor's opportunistic resources;
- tolerance to resource failures;
- options on branching, searching and presolving similar to commercial software such as CPLEX;
- the ability to process both industrial standard MPS format and GAMS model as input;

The remainder of this paper is organized as follows. Section 2 describes the BB algorithm and the components of FATCOP that are implemented to ensure the BB algorithm generates a reasonable search tree. Section 3 introduces Condor-PVM parallel programming framework and the design of our parallel implementation. In section 4, we present some numerical results that exhibit important features of FATCOP.

2 Components of Sequential Program

For expositional purposes, we review the basic sequential implementation of a BB MIP solver. A MIP can be stated mathematically as follows:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \leq b \\ & l \leq x \leq u \\ & x_j \in Z \qquad \forall j \in I \end{array}$$

where Z denotes the integers, A is an $m \times n$ matrix, and I is a set of distinguished indices identifying the integer variables.

Most integer programming textbooks [1, 20] describe the fundamental branch-and-bound algorithm for the above MIP problem. Basically, the method explores a binary tree of subproblems. The key steps of BB are summarized below and the algorithm flowchart can be found as Figure 1.

Algorithm *Branch-and-Bound:*

Step 0. *Initialization:*
Set iteration count $k = 0$ and incumbent (or upper bound of the MIP) $Z^* = \infty$. Put the initial problem in the *work pool* containing subproblems that are not examined yet.

Step 1. *Branching:*
Among the remaining subproblems in the work pool, select one according to some search order. Among the integer-restricted variables that have a noninteger value in the optimal solution for the LP relaxation of this subproblem, choose one according to some branching rule to be the branching variable. Let x_j be this variable and denote by x_j^* its value in the aforementioned solution. Branch from the node for the subproblem to create two new subproblems by adding the respective constraints $x_j \leq \lfloor x_j^* \rfloor$ and $x_j \geq \lfloor x_j^* \rfloor + 1$.

Step 2. *Bounding:*
For each new subproblem, obtain its bound by applying the simplex method (or the dual simplex method when reoptimizing) to its LP relaxation and use the value of Z_{LP} for the resulting optimal solution. The minimum value of such bounds associated with subproblems in the work pool is referred to as lower bound of the MIP.

Step 3. *Fathoming:*
For each new subproblem, apply the three fathoming tests given below, and discard those subproblems that are fathomed by any of the tests.
Test 1: Its bound $Z_{LP} \geq Z^*$.
Test 2: Its LP relaxation has no feasible solutions.

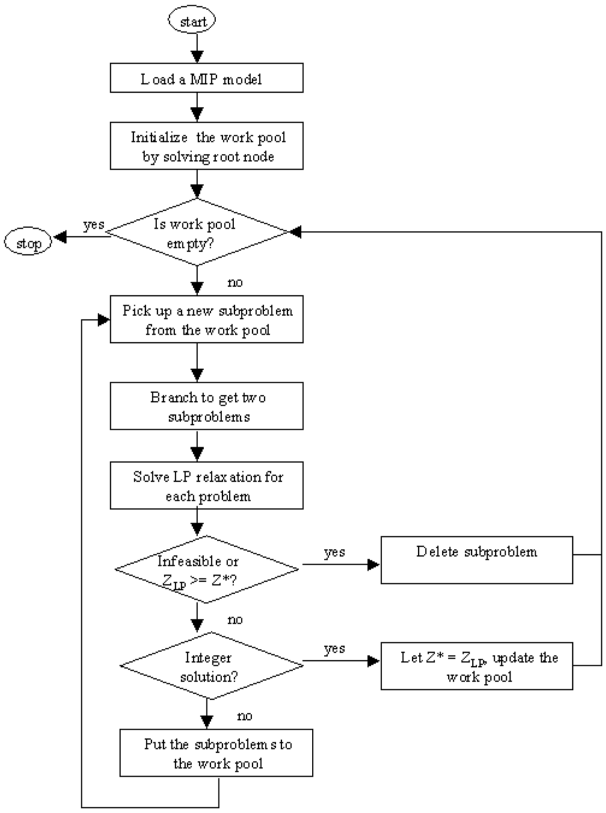

Figure 1: Branch-and-bound algorithm

Test 3: The optimal solution for its LP relaxation has integer values for the integer-restricted variables. If this solution is better than the incumbent, it becomes the new incumbent and Test 1 is reapplied to all unfathomed subproblems with the new Z^*.

Step 4. *Optimal test:*
Stop when there are no remaining subproblems. The current incumbent is optimal. Otherwise, go to step 1.

The above description makes it clear that there are various choices to be made during the course of the algorithm. We hope to find robust strategies that work well on a wide variety of problems. The reminder of this section describes the algorithm components and FATCOP's interfaces.

2.1 Branching rules

If there are many fractional variables in the solution to the LP relaxation, we must choose one variable to be the branching variable. Because the effectiveness of the branch and bound method strongly depends on how quickly the upper and lower bounds converge, we would like to branch on a variable that will improve these bounds.

Several reasonable criteria exist for selecting branching variables. FATCOP currently provides three variable selection options: pseudocost, maximum integer infeasibility and minimum integer infeasibility. Maximum integer infeasibility will generate two branches that are more likely to differ from the current solution than other branch alternatives. The underlying assumption is that the objective function values are more likely to degrade on both branches. Hence such a branch would represent an influential variable. Minimum integer infeasibility will generate one branch very similar to the current solution and one branch very different from the current solution. The underlying assumption is that the similar branch is where a solution lies and that the different branch will prove uninteresting [1]. Pseudocost attempts to estimate the change rate in objective function value associated with a particular branch. Since the pseudocost method is widely used and known to be efficient [16], we set it as the default branching strategy and briefly describe the method here.

We associate two quantities ϕ_j^- and ϕ_j^+ with each integer variable x_j that attempts to measure the per unit decrease in objective function value if we fix x_j to its rounded down value and rounded up value, respectively. Suppose that $x_j = \lfloor x_j \rfloor + f_j$, with $0 < f_j < 1$. Then by branching on x_j, we will estimate a decrease of $D_j^- = \phi_j^- f_j$ on the down branch and a decrease of $D_j^+ = \phi_j^+(1 - f_j)$ on the up branch . The way to obtain the objective change rate ϕ_j^- and ϕ_j^+ is to use the observed change in objective function value:

$$\phi_j^- = \frac{Z_{LP}^- - Z_{LP}}{f_j} \quad \text{and} \quad \phi_j^+ = \frac{Z_{LP}^+ - Z_{LP}}{1 - f_j}$$

where Z_{LP} is the LP relaxation solution value, and Z_{LP}^+ and Z_{LP}^- are the LP relaxation solution values at up and down branches respectively. Note that the above discussion is for a particular node.

In the course of the solution process, the variable x_j may be branched on many times. The pseudocosts are updated by averaging the values from all x_j branches. The remaining issue is to determine to what values should the pseudocost be initialized. The question of initialization is important, since the branching decisions made at the top of the tree are the most crucial. If similar MIP problems have been solved before, we can use the previous results to initialize pseudocosts for integer variables. If previous results are not available, we can explicitly solve LPs to compute pseudocosts for candidate branching variables which have not been branched yet. Our computational experience show that it is expensive to compute these values explicitly, especially in case that the number of integer variables is large. Therefore, we simply initialize the pseudocost x_j with its corresponding coefficient in the objective function c_i. When taking a branching decision for subproblem P, the algorithm calculates a score s_j for each x_j as suggested in [9]:

$$s_j = \alpha_0 p_j + \alpha_1 D_j^+ + \alpha_2 D_j^-;$$

where p_j is the user specified priority for x_j, $p_j = 0$ if the priority is not specified. α_0, α_1,α_2 are some constants. α_0 usually is very big so that user specified priorities are predominant. Normally the branching candidate variable with highest priority will be selected.

2.2 Searching rule

FATCOP provide five options for selecting a node from the remaining nodes:

1. Depth-first: The depth-first rule selects the node that was created most recently. The depth-first rule has the advantage that it needs very small memory and usually finds a feasible solution quickly.

2. Best-first: The best-first rule, as its name suggests, always selects the current node to be the one whose LP solution is the smallest. Since the best-first rule always select the most attractive subproblem, it will improve the algorithm's convergence speed.

3. Pseudocost-estimation: An estimation for the optimal value of an integer solution for a node can be obtained from its LP solution and its branching candidates' pseudocost:

$$E = Z_{LP} + \sum_{j \in \{\text{all branching candidates}\}} \min\{D_j^+, D_j^-\}$$

   The pseudocost-estimation strategy selects the node with the best estimation from the remaining nodes. Like depth-first search, it requires small memory and usually can find good feasible solutions quickly. Therefore, pseudocost-estimation is often used to attack complex MIP problems when the provable optimal solution is hard to obtain.

4. A mixed strategy of 1 and 2: This strategy expands the subproblems in the best-first order, with an initial depth-first phase. The mixed strategy takes the advantage of quickly finding a feasible solution by depth first, and fastest convergence by best first. The central issue of this strategy is when the program should switch from 1 to 2. Eckstein [9] suggested to compare the lower bound and upper bound of a MIP problem, i.e. the BB algorithm pursues depth-first search as long as the the upper bound and lower bound differs at least some fixed threshold percentage. Afterwards it expands problems with best-first. When designing FATCOP, we propose to keep track of the information about the number of node evaluations over which the best integer solution has not been updated. FATCOP switches searching strategy from 1 to 2 after this number exceeds a pre-specified fixed number. In practice we found that it was useful to perform a short trial solve test on a MIP problem using depth-first strategy. If we observe that the incumbent value keeps unchanged for long time after it was updated for $\tilde{U}$ times in the trial solve process, we then can let the program switch searching rule at that point when we solve this problem using the mixed searching strategy. If $\tilde{U}$ is set to 0, the program will use pure best-first strategy, while $\tilde{U} = \infty$ implies a pure depth-first strategy.

5. A mixed strategy of 1 and 3: This strategy is similar to strategy 4, but starts the algorithm with pseudocost-estimation search first. Since pseudocost-estimation often finds better solutions than depth first does, strategy 5 is set as default searching strategy for FATCOP.

2.3 Preprocessing

Preprocessing refers to a set of simple reformulations performed on a problem instance to enhance the solution process. FATCOP incorporates some preprocessing techniques to identify infeasibility and redundancies, to tighten bounds on variables, and to improve coefficients of constraints. Preprocessing may reduce the size of a MIP problem as well as the integrality gap, i.e., the difference between the optimal solution value and its LP relaxation. More techniques for preprocessing and probing can be found in [24].

Without loss of generality we assume that the inequality currently under consideration is of the form:

$$\sum_j a_j x_j \leq b$$

Define:

l_j	The lower bound of variable x_j;
u_j	The upper bound of variable x_j;
L_{min}	The minimum possible value for left hand side of the constraint ;
L_{max}	The maximum possible value for left hand side of the constraint ;
L^k_{min}	The minimum possible value for left hand side of the constraint without the term $a_k x_k$;
L^k_{max}	The maximum possible value for left hand side of the constraint without the term $a_k x_k$;

The following techniques may allow problem reduction:

1. Simple presolving methods:

   - remove empty row or column
   - check infeasible or fixed variable: $l_j > u_j$ or $l_j = u_j$
   - remove singleton row and modify the corresponding bounds.

2. Identification of infeasibility:
$$L_{min} > b$$

3. Identification of redundancy:
$$L_{max} \leq b$$

4. Improvement of bounds: For each variable x_k in the constraint, we have:
$$L^k_{min} + a_k x_k \leq \sum_j a_j x_j \leq b$$
   Then:
$$a_k x_k \leq b - L^k_{min}$$
   If $a_k > 0$,
$$x_k \leq \min\{(b - L^k_{min})/a_k, u_k\}$$
   If $a_k < 0$,
$$x_k \geq \max\{(b - L^k_{min})/a_k, l_k\}$$
   If x_k is an integer constrained variable, the new upper and lower bounds should be rounded down and up.

5. Improvement of coefficients: Define $\delta = b - L^k_{max}$. If x_k is a binary variable and $\delta > 0$. Both a_k and b can be reduced by δ. One can easily checks the validity by setting x_k to 0 and 1 respectively.

At the root node, FATCOP analyzes every row of the constraint matrix using the above techniques. If, after processing, some variables are fixed or some bounds are improved, the process is repeated until no further model reduction occurs.

2.4 Reduced Cost Checking

After solving subproblem P, one checks the LP reduced costs of all integer variables x_j that are not basic. If the absolute value of such a reduced cost exceeds $Z^* - Z_{LP}$, then x_j is fixed at its present value in all of P's descendents. The reduced cost of variable x_j represents the objective function value changing rate with respective to x_j. Therefore any descendant solution with a different but still integral value of x_j would be bigger than Z^*, hence would necessarily be fathomed.

2.5 Interfaces

FATCOP accepts industrial standard MPS input as well as GAMS models. MPS [19] input format was originally introduced by IBM to express linear and integer programs in a standard way. All commercial LP and MIP software accept this format.

GAMS [4] is a high-level modeling system for mathematical programming. It takes as input a description of a mathematical program in a form that people find reasonably natural and convenient, and allows the solution output to be viewed in similar terms. GAMS is tailored for complex, large scale modeling applications, and allows users to build large maintainable models that can be adapted quickly to new situations. We interfaced FATCOP to GAMS using the GAMS IO library. With the GAMS interface, FATCOP is able to exploit larger classes of MIP problems.

3 Condor-PVM parallel implementation of FATCOP

In this section we first give a brief overview of Condor, PVM and the Condor-PVM parallel programming environment. Then we discuss the parallel scheme we selected for FATCOP and the differences between normal PVM and Condor-PVM programming. At the end of the section, we present a detailed implementation of FATCOP.

3.1 Condor-PVM Parallel Programming Environment

Nowadays heterogeneous clusters of workstations are becoming an important source of computing resources. Two approaches have been addressed to make effective use of such resources. One approach provides efficient resource management by allowing users to run their jobs on idle machines that belong to somebody else. Condor, developed at University of Wisconsin-Madison, is one such system. It monitors the activity on all participating machines, placing idle machines in the Condor "pool". Machines are then allocated from the pool when users send job requests to Condor. Machines enter the pool when they become idle, and leave when they get busy, i.e. the machines owner returns. When an executing machine becomes busy, the job running on this machine is initially suspended in case the executing machine becomes idle again within a timeout period. If the executing machine remains busy then the job must be migrated to another idle workstation in the pool or returned to the job queue. For a job to be restarted after migration to another machine a checkpoint file that allows the exact state of the process to be re-created is required. This design feature ensures the eventual completion of a job. There are various priority orderings used by Condor for determining which jobs and machines are matched at any given instance. Based on these orderings, sometimes running jobs may be preempted to allow higher priority jobs to run instead. Condor is freely

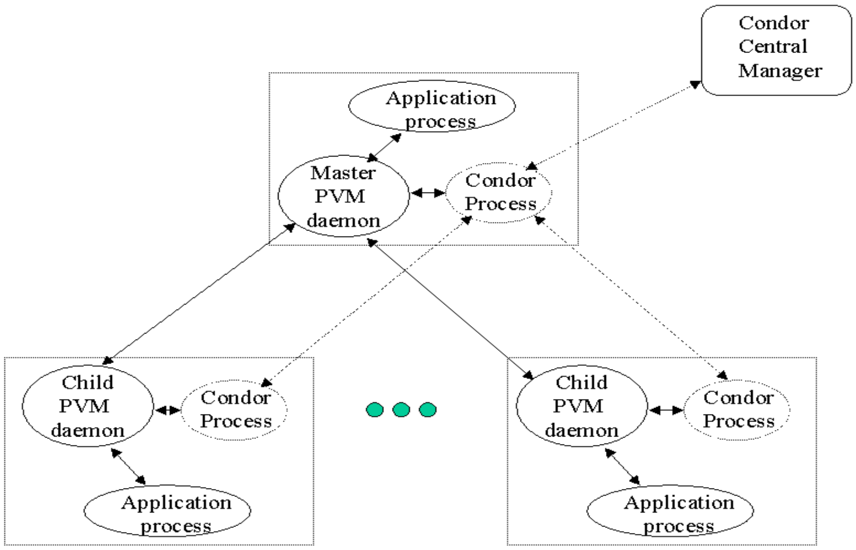

Figure 2: Architecture of Condor-PVM

available and has been used in a wide range of production environments for more than ten years.

Another approach to exploit the power of workstation cluster is from the perspective of parallel programming. Research in this area has developed message passing environments allowing people to solve a single problem in parallel using multiple resources. One of the most widely used message passing environments is PVM that was developed at the Oak Ridge National Laboratory. PVM transparently handles all message routing, data conversion and task scheduling across a network of incompatible computer architectures. The goal of PVM is to make programming for a heterogeneous collection of machines straightforward. The PVM system is composed of two parts. The first part is a daemon which resides on all the computers making up the virtual machine. A virtual machine is created in each computer where PVM applications run on. The second part of the system is the PVM library. It contains user-callable routines for message passing, process spawning, virtual machine modification and task coordination. A similar message passing environment is MPI [14]. Both systems center around a message-passing model, providing point-to-point as well as collective communication between distributed processes. The primary differences between the systems lie in their views of the parallel computer. PVM's design centers around the idea of a *virtual machine*. This virtual machine is very general and can encompass a nearly arbitrary collection of computing resources, from desktop workstations to multiprocessors to massively parallel homogeneous supercomputers. To provide uniform access to such a complex set of resources, PVM provides process control and resource management functions that allow spawning and termination of arbitrary processes and the addition and deletion of hosts at runtime. This flexibility, however, comes at the expense of communication performance. MPI foregoes this flexibility in favor of optimal communication performance.

The development of resource management system and message passing environment has been independent of each other for many years. Unfortunately, existing resource management systems usually cannot bring resources together and message passing environments are not able to use resources efficiently. Researchers at the University of Wisconsin-Madison recently developed a parallel programming framework which interfaces Condor and PVM [22, 6]. The reason to select PVM instead of MPI is that the implementation of MPI has no concept of process control, hence cannot handle resource addition and retreat in a opportunistic environment. Figure 2 shows the architecture of Condor-PVM. There are three processes on each machine running a Condor-PVM application: PVM daemon, Condor process and user application process. The Condor-PVM framework still relies on the PVM primitives for application communication, but provides resource management which is suited to the opportunistic environment provided by Condor. Each PVM daemon has a Condor process associated with it, acting as the *resource manager*. The Condor process interacts with PVM daemon to start tasks, send signals to suspend, resume and kill tasks, and receive process completion information. The Condor process running on the master machine is special. It communicates with Condor processes running on the other machines, keeps information about the status of the machines and forwards resource requests to Condor central manager. This Condor process is called the *global resource manager*. When a Condor-PVM application asks for a host, the global resource manager communicates with Condor central manager to schedule a new machine. After Condor grants a machine to the application, it starts a Condor process (resource manager) and PVM daemon on the new machine. If a machine needs to leave the pool, the resource manager will send signals to the PVM daemon to suspend tasks. The master user application is notified of that via normal PVM notification mechanisms.

Compared with a conventional dedicated environment, the Condor-PVM opportunistic environment has the following characteristics:

1. There usually are a large amount of heterogeneous resources available for an application, but in each time instance, the amount of available resources is random, dependent on the status of machines managed by Condor. The resources are also competed for by owners and other Condor users.

2. Resources used by an application may exit during its life cycle.

3. The execution order of components in an application is highly non-deterministic, leading to different solution and execution times.

Therefore a good Condor-PVM application should be tolerant to loss of resources (host suspension and deletion) and dynamically adaptive to the current status of Condor pool in order to make effective use of opportunistic resources.

3.2 Parallel Scheme for FATCOP

Three main approaches in designing parallel BB algorithms are summarized in [13] . Approach 1 consists of solving LP relaxations in parallel for each subproblem to accelerate the execution. This approach uses parallel LP solvers and the order of subproblems generated during the execution of BB algorithm is fixed. Approach 2 introduces parallelism when building the BB tree. It performs bounding operations on several subproblems simultaneously. This approach may affect the order of subproblems generated during the expansion of the BB tree. Hence more or less subproblems could be evaluated by the parallel program compared with its sequential version. Such phenomena are known as *search anomalies*. Approach 3 consists of building several BB trees in parallel. The trees are built up by applying different operations such as branching and searching. The information generated when building one tree can be used for the construction of another. We choose the second approach to implement FATCOP, since it is widely used and appropriate for the implementation under Condor-PVM framework.

One more concern about the parallel BB algorithm is central versus decentralized control. Recall in section 2 we used the notion of work pool, which is a memory location where processors find and store the subproblems that are not yet examined yet. The centralized parallel BB program has a single work pool and is usually implemented by using the master-slave paradigm. One processor, called the master manages the work pool, and sends pieces of work out to other processors, called slaves, that do some computation, and send the results back to the master. This centralized parallel scheme suffers from a "bottleneck". When using a large number of processors, the master can become a bottleneck in processing the returned information, thus keeping slaves idle for large amounts of time. In decentralized programs, each processor has its own local work pool. Sometimes these kind of programs are called multi-pool programs, while centralized programs are called single-pool programs. The central issue of this parallel scheme is load balancing. The parallel algorithm must make sure that some processors do not run out of work while others have a large amount of tasks.

The Condor-PVM environment allows us to design both multi-pool and single-pool programs, since each machine in Condor pool has its own memory location. However, multi-pool programs do not lend themselves to the Condor opportunistic environment. In such an environment, any of the machines could be preempted and therefore disappear at any moment. The pool of problems associated with the disappeared machine will be lost. A multi-pool program usually allows communication between processors to exchange subproblems. This feature makes it even harder for the program to keep track of the lost problems. On the other hand, the master-slave model, as an example of single-pool program, can handle resource retreat well. The idea is that the master keeps track of which subproblem has been sent to each slave, and does not actually remove the subproblem out of the work pool. All the subproblems which are sent out are marked. If the master is then informed that a slave has disappeared, it unmarks the subproblems assigned to that slave.

The remaining design issue is how to use the opportunistic resources provided by Condor to adapt to changes in number of available resources. The changes include newly available machines, machine suspension and resumption and machine failure. We hope to find a good design that can handle changes in the size of the virtual machine naturally. In a conventional dedicated environment, a parallel application usually is developed for running with a fixed number of processors and the solution process will not be started until the required number of processors are obtained and initialized. In Condor's opportunistic environment, doing so may cause a serious delay. In fact the time to obtain the required number of new hosts from Condor pool can be unbounded. Therefore we implement FATCOP in such a way that the solution process starts as soon as it obtains a single host. The solver then attempts to acquire new hosts as often as possible. At the beginning of the program, FATCOP places a number of requests for new host in Condor. Whenever it gets a host, it requests a new host. Thus, in each period between when Condor assigns a machine to FATCOP and when the new host request is received by Condor, there is at least one "new host" request from FATCOP waiting to be processed by Condor. This greedy implementation makes it possible for a FATCOP job to collect a significant amount of hosts during its life cycle.

3.3 Differences between PVM and Condor-PVM programming

Regular PVM and Condor-PVM are binary compatible with each other. The same binary which runs under regular PVM will run under Condor, and vice-versa. However there exist some run time differences between regular PVM and Condor-PVM. The most important difference is the concept of machine class. In a regular PVM application, the configuration of hosts that PVM combines into a virtual machine usually is defined in a file, in which host names have to be explicitly given. Under the Condor-PVM framework, Condor selects the machines on which a job will run, so the dependency on host names must be removed from an application. Instead the applications must use class names. Machines of different architecture attributes belong to different machine classes. Machine classes are numbered 0, 1, etc. and hosts are specified through machine classes. A machine class is specified in the submit-description file that is submitted to Condor.

Another difference is that Condor-PVM has "host suspend" and "host resume" notifications in addition to "host add", "host deletion" and "task exit" notifications that normal PVM has. When Condor detects activity of a workstation owner, it suspends all Condor processes running there rather than killing them immediately. If the owner remains for less than a pre-specified cut-off time, the suspended processes will resume. To help an application to deal with this situation, Condor-PVM make some above extensions to PVM's notification mechanism.

The last difference is that adding a host is non-blocking in Condor-PVM. When a Condor-PVM application requests a new host be added to the virtual machine, the request is sent to Condor. Condor then attempts to schedule one from the pool of idle machines. This process can take a significant amount of time. If there are no machines available in Condor's pool, the delay can be longer. Therefore, Condor-PVM handles requests for new host asynchronously. The application can start other work immediately after it sends out a request for new host. It then uses the PVM notification mechanism to detect when the request has completed. People interested in developing Condor-PVM applications should pay attention to these differences. The documentation about the differences and a sample example code can be found in [6] and Condor manual website at

http://www.cs.wisc.edu/condor/

FATCOP is first developed as a regular PVM application, and modified to exploit Condor-PVM.

3.4 Parallel Implementation of FATCOP

FATCOP consists of two separate programs: the master program and the slave program. The master program runs on the machine on which the job was submitted to Condor. This machine is supposed to be stable for the life of the run, so it is generally the machine owned by the user. The design of FATCOP makes the program tolerant to any type of failures for workers, but if the machine running the master program crashes due to either system reboot or power outage, the program will be terminated. To make FATCOP tolerant even of the master's failure, the master program writes information about subproblems in the work pool periodically to a log file on the disk. Each time a FATCOP job is started by Condor, it reads in the MIP problem as well as the log file that stores subproblem information. If the log file does not exist, the job starts from the top of the search tree. Otherwise, it is warm started from some point in the search process. The work pool maintained by the master program

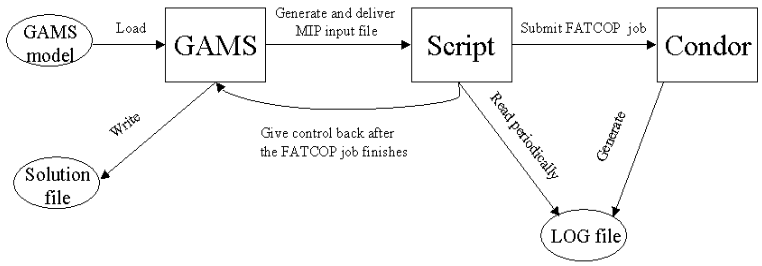

Figure 3: Interactions among Condor, FATCOP and GAMS

has copies for all the subproblems that were sent to the slaves, so the master program is able to write complete information about the BB process to the log file.

Copies of the slave program run in the machines selected by Condor. The number of copies of running slave programs changes over time during the execution of a FATCOP job.

3.4.1 The Master Program

FATCOP can take both MPS and GAMS models as input. The interactions among Condor, FATCOP and GAMS are described as follows. A user starts to solve a GAMS model in the usual way from the command line. After GAMS reads in the model, it generates an input file. Control then is given to the user's control file, which usually is a shell or PERL script. The script generates a Condor submit-description file and submits the job to Condor. The submit-description file specifies the program name, input file name, requirement on machines' architecture, operating system and memory etc. After submitting the job the script reads a log file periodically until the submitted job is finished by Condor. The log file is generated by Condor and records the status of the finished and executing jobs. Finally the script gives control back to GAMS. GAMS then reports the solutions to the user. This process is depicted in Figure 3.

The MIP model is stored globally as an LP and integrality constraints. The master program first solves the LP. If it is infeasible or the solution satisfies the integrality constraints, the master program stops. Otherwise, it starts a sequential MIP solve process until there are N subproblems in the work pool. N is a pre-defined number, which has a default value and can be modified by users. This process is based on the observation that using parallelism as soon as few subproblems become available may not be a good policy, since doing so may expand more nodes compared to the sequential algorithm. In order to use all processors as

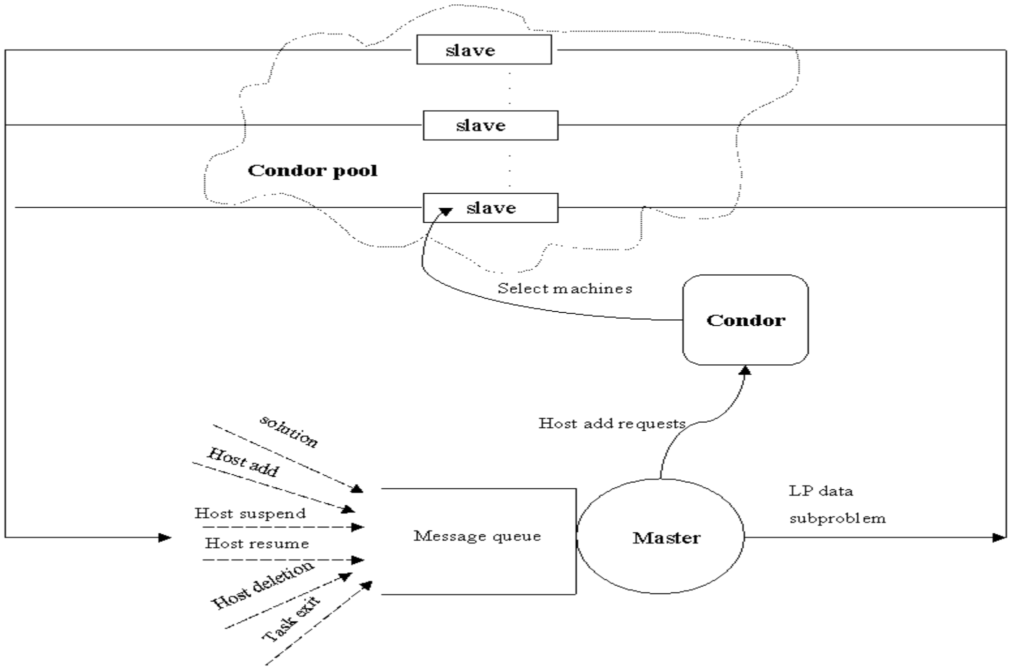

Figure 4: Message passing inside FATCOP

soon as possible while avoiding giving them subproblems that only have a small chance of leading to an optimal solution, we let the master perform a sequential algorithm up to a point where N subproblems are available. The work pool consists of solved subproblems. Associated with each subproblem is the LP relaxation solution and value, modified bound information for the integer variables, pseudocosts used for searching and an optimal basis, which is used for warm starting the simplex method. The subproblems in the work pool are multi-indexed by bound, pseudocost-estimation and the order in which they entered the pool. The indices correspond to different searching rules: best-first, pseudocost-estimation and depth-first.

Following the initial subproblem generation stage, the master program sends out a number of requests for new hosts. It then sits in a loop that repeatedly does message receiving. The master accepts several types of messages from workers. The messages passing within FATCOP are depicted in Figure 4.

After all workers have sent solutions back and the work pool becomes empty, the master program kills all workers and exits itself.

Host Add Message. After the master is notified of getting a new host, it spawns a child process there and sends an LP copy as well as a subproblem to the new child process. The subproblem is marked in the work pool, but not actually removed from it. Thus the master is capable of recovering from several types of failures. For example, the spawn may fail. Recall that Condor takes the responsibility to find an idle machine and starts a PVM daemon on it. During the time between when the PVM daemon was started and the message received by master program, the owner of the selected machine can possibly reclaim it. If a "host add" message was queued waiting for the master program to process other messages, a failure for spawn becomes more likely.

The master program then sends out another request for a new host if the number of remaining subproblems is at least twice as many as the number of workers. The reason for not always asking for new host is that the overhead associated with spawning processes and initializing new hosts is significant. Spawning a new process is not handled asynchronously by Condor-PVM. While a spawn request is processed, the master is blocked. The time to spawn a new process usually takes several seconds. Therefore if the number of subproblems in the work pool drops to a point close to the number of workers, the master will not ask for more hosts. This implementation guarantees that only the top 50% "promising" subproblems considered by the program can be selected for evaluation. Furthermore, when the BB algorithm eventually converges, this implementation prevents the program from asking for excess hosts. However, the program must be careful to ensure that when the ratio of number of remaining subproblems to number of hosts becomes bigger than 2, the master restarts requesting host additions.

Solution Message. If a received message contains a solution returned by a worker, the master will permanently remove the corresponding subproblem from the work pool that was marked before. It then updates the work pool using the received LP solutions. After that, the master selects one subproblem from the work pool and sends it to the worker which sent the solution message. The subproblem is marked and stays in the work pool for failure recovery. Some worker idle time is generated here, but the above policy typically sends subproblems to workers that exploit the previously generated solution.

Host Suspend Message. This type of messages informs the master that a particular machine has been reclaimed by its owner. If the owner leaves within 10 minutes, the Condor processes running on this machine will resume. We have two choices to deal with this situation. The master program can choose to wait for the solutions from this host or send the subproblem currently being computed in this host to another worker. Choosing to wait may save the overhead involved in solving the subproblem. However the waiting time can be as long as 10 minutes. If the execution time of a FATCOP job is not significantly longer than 10 minutes, waiting for a suspended worker may cause a serious delay for the program. Furthermore, the subproblems selected from the work pool are usually considered "promising". They should be exploited as soon as possible. Therefore, if a "host suspend" message is received, we choose to recover the corresponding subproblems in the work pool right away. This problem then has a chance to be sent to another worker shortly. If the suspended worker resumes later, the master program has to reject the solutions sent by it in order that each subproblem is considered exactly once.

Host Resume Message. After a host resumes, the master sends a new subproblem to it. Note that the master should reject the first solution message from that worker. The resumed worker picks up in the middle of the LP solve process which was frozen when the host was suspended. After the worker finishes solving the LPs, it sends the solutions back to the master. Since the associated subproblem had been recovered when the host was suspended, these solutions are redundant, hence should be ignored by the master.

Host Delete/ Task Exit Message. If the master is informed that a host is removed from the PVM virtual machine or a process running on a host is killed, it recovers the corresponding subproblem from the work pool and makes it available to other workers.

Name	#rows	#columns	#nonzeros	#integers
AIR05	426	7195	52121	7195
BELL5	91	104	266	58
BLEND2	274	353	1409	264
FIBER	363	1298	2944	1254
MAS76	12	151	1639	150
MISC07	212	260	8619	259
QIU	1192	840	3432	48
STEIN45	331	45	1034	30
VOD1	107	306	1207	303
PROD1	211	301	10501	200

Table 1: Summary of test problems

3.4.2 Slave Program

The slave program first receives a LP model from the master, then sits in an infinite loop to receive messages from the master. The messages from the master consist of the modified bound information about the subproblem P, the optimal basis to speed up the bounding operation, and the branching variable which is used to define the "up" and "down" children $P+$ and $P-$. The slave performs two bounding operations on $P+$ and $P-$ and sends the results back to the master. The slave program is not responsible for exiting its PVM daemon. It will be killed by the master after the stopping criteria is met.

4 Computational Experience

One of FATCOP's goals is fault tolerance, i.e. solving MIP problems correctly using opportunistic resources. The other design purpose is to let FATCOP be adaptive to changes in available sources provided by Condor in order to achieve maximum possible parallelism. Therefore the principle measures we use when evaluating FATCOP are *correctness* of solutions, and *adaptability* to changes in resources. *Execution time* should be another important performance measure, but it is affected by many random factors and heavily dependent on the availability of Condor's resources. For example a FATCOP job, that can be finished in one hour at night, may take 2 hours to finish during the day because of the high competition for the resources. In general, FATCOP should have shorter execution time than its serial version for long-running MIP problems. This is confirmed by the results presented in this section.

FATCOP has been tested on a set of problems drawn from MIPLIB [18],an electronically available library of both pure and mixed integer programs arising from real applications. The selected problems have relatively large search trees, so that some parallelism can be exploited. Since Condor-PVM incurs some additional managerial and communication costs to a FATCOP job, some problems with small search trees can be solved faster by the FATCOP sequential program than its parallel version. We first used CPLEX [7] with default settings to solve the problems in MIPLIB. We then picked 8 problems which cannot be solved by CPLEX within 10,000 nodes. In order to demonstrate that FATCOP can also solve GAMS

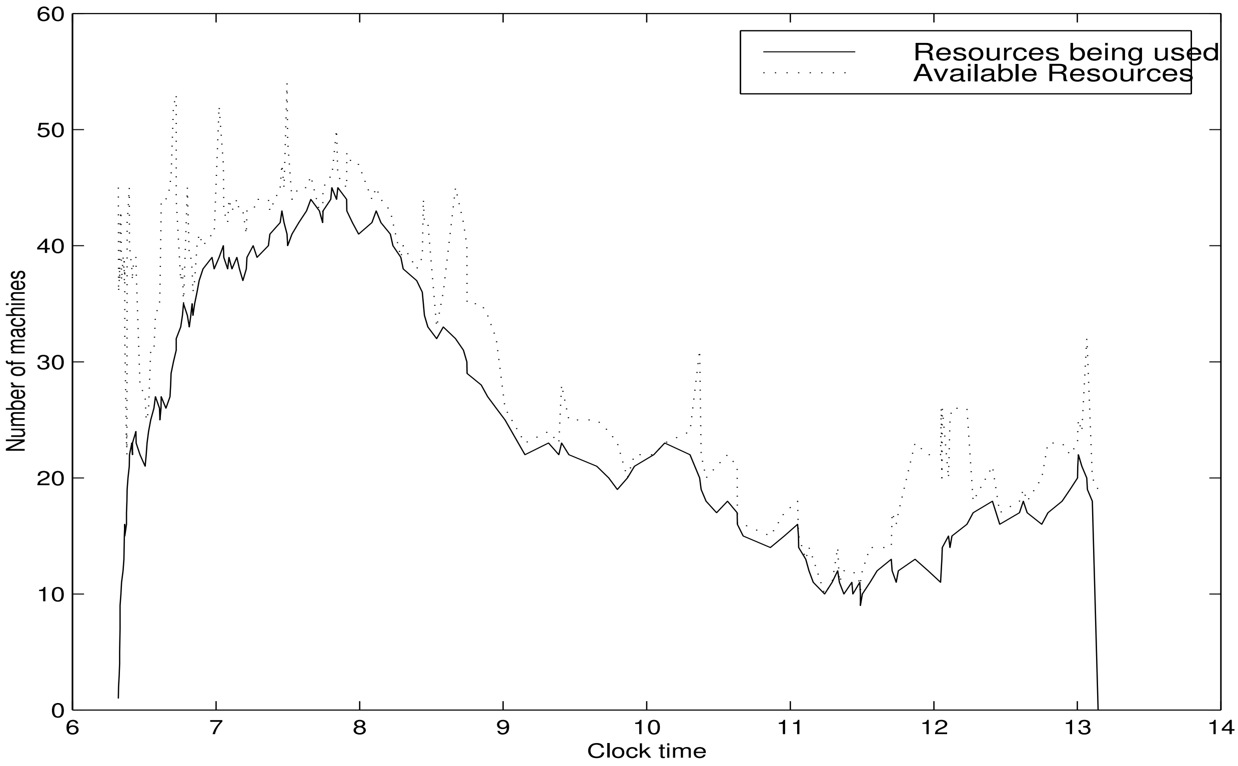

Figure 5: Resource utilization for one run of FATCOP

models, two problems formulated using GAMS are included in the test set. One problem *vod3* is an application to video on demand system design [8]. The other problem *prod1* is an application to product design [25]. These two problems are also insoluble in 10,000 nodes by the CPLEX default MIP solver. The sizes of the problem instances are shown in Table 1.

4.1 Resource Utilization

In Wisconsin's Condor pool there are more than 100 machines in our desired class. Such large amounts of resources make it possible to solve MIP problems with fairly large search trees. However the available resources provided by Condor change as the status of participating machines change. Figure 5 demonstrates how FATCOP is able to adapt to Condor's dynamic environment. We submitted a FATCOP job to solve *air05* at 6:30am. The job was finished by Condor at about 13:10. Each time a machine was added or suspended, the program asked Condor for the number of idle machines in our desired machine class. We plot the number of machines used by the FATCOP job and the number of machines available to the job in Figure 5. In the figure, time goes along the horizontal axis, and the number of machines is on the vertical axis. The solid line is the number of working machines and dotted line is the number of available machines which includes idle machines and working machines used by our FATCOP job. At the start, there were many idle machines in Condor pool. The job quickly got about 20 machines and eventually collected more than 40 machines with a speed of roughly one new resource every minute. However it was not able to acquire many machines after 7:00am and lost a large amount of machines between 8:00 to 9:00. There were some newly available resources at 8:30 and 10:00, but they became unavailable again

Run	Starting time	Duration	E_{avg}	Number of suspensions
1	06:30	6.7 hrs	28	45
2	08:15	6.8 hrs	31	51
3	14:55	6.1 hrs	46	43
4	15:02	6.1 hrs	42	50
5	20:10	5.9 hrs	61	39
6	21:42	6.0 hrs	51	58

Table 2: Average number of machine and suspensions for 6 FATCOP runs

quickly, either reclaimed by owners or scheduled to other users by Condor. At about 12:00, another group of machines became available and stayed idle for a relatively long time. The job thus acquired some additional machines during that time. In general, the number of idle machines in Condor pool had been kept at a very low level during life cycle of the FATCOP job except during the start-up phase. When the number of idle machines stayed high for some time, FATCOP was able to quickly increase the size of its virtual machine. We believe these observations exhibit that FATCOP can utilize opportunistic resources very well.

To get more insight about utilization of opportunistic resources by FATCOP, we define the average number of machines used by a FATCOP job E_{avg} as:

$$E_{avg} = \frac{\sum_{k=1}^{E_{max}} k\tau_k}{T},$$

where τ_k is the total time when the FATCOP job has k workers, T is the total execution time for the job, E_{max} is the number of available machines in the desired class. We ran 6 replications on *air05*. The starting time of these runs is distributed over a day. In Table 2 we record the average number of machines the FATCOP job was able to use and number of machines suspended during each run. The first value shows how much parallelism the FATCOP job can achieve and the second value indicates how much additional work had to be done. In general the number of machines used by FATCOP is quite satisfactory. For run 5, this value is as high as 61 implying that on average FATCOP used about 60% of the total machines in our desired class. However, the values vary greatly due to the different status of the Condor pool during different runs. In working hours it is hard to acquire machines because many of them are used by owners. After working hours and during the weekend only other Condor user's are our major competitors. As expected FATCOP lost machines frequently during the daytime. However during the runs at night FATCOP also lost many machines. It is not surprising to see this, because the more machines FATCOP was using, the more likely it would lose some of them.

4.2 Fault tolerance

For all the test problems in Table 1 except *bell5*, FATCOP was configured to use pseudocost branching and pseudocost-estimation based mixed searching strategy (strategy 5). It turned out that depth-first based mixed strategy (strategy 4) worked best for *bell5*. We performed a trial solve for each problem to decide the value of $\tilde{U}$ as introduced in section 2.2. Recall $\tilde{U} = \infty$ implies a pure pseudocost-estimation search, and $\tilde{U} = 0$ implies a pure best-first search. The

Name	Solution gap(%)	Provable optimal?	Execution time	Tree size	$\tilde{U}$
AIR05	2	no	8 hrs	1,615	∞
BELL5	0	yes	5.1hrs	71,823	1
BLEND2	0	yes	9.9 min.	1,548	0
FIBER	57	no	8 hrs	120,432	∞
MAS76	0	no	8 hrs	162,176	∞
MISC07	0	yes	1.9 hrs	10,785	6
QIU	0	no	8 hrs	22,004	3
STEIN45	0	yes	3.5 hrs	52,557	1
VOD3	0	yes	30.2 min	6,201	7
PROD1	3	no	8 hrs	32,524	∞

Table 3: Results obtained by the FATCOP sequential solver

Name	Execution time	Tree size	E_{avg}	# of suspensions
AIR05	6.0 hrs	4,142	46	51
BELL5	41.8 mins	39,026	20	19
BLEND2	11.0 mins	2,589	10	0
FIBER	4.5 hrs	231,887	52	81
MAS76	1.2 hrs	420,418	38	5
MISC07	1.0 hrs	9,491	22	15
QIU	5.4 hrs	66,180	57	51
STEIN45	1.3 hrs	55,110	41	17
VOD3	19.5 min	6,540	11	1
PROD1	5.5 hrs	98,227	48	49

Table 4: Average results obtained by the FATCOP parallel Condor-PVM solver for 3 replications (all instances are solved to optimality)

	Execution time		Tree size	
	sequential	parallel	sequential	parallel
BELL5	5.1 hrs	41.8 mins	71,823	39,026
BLEND2	9.9 min.	11.0 min.	1,548	2,598
MISC07	1.9 hrs	1.0 hrs	10,785	9,491
STEIN45	3.5 hrs	1.3 hrs	52,557	55,110
VOD3	30.2 min	19.5 min	6,201	6,540

Table 5: Comparison between the FATCOP sequential and parallel solver for the problems solved to optimality by both

parameters for computing scores of branching variables are set as follows: $\alpha_0 = 100$,$\alpha_1 = 1$, and $\alpha_2 = 1$. The MIPLIB files do not provide branching priorities, so the value of α_0 is irrelevant to problems from MIPLIB. However the priority information about integer variables is critical to solve *vod* in a reasonable amount of time.

We first solved the problems in Table 1 using the FATCOP sequential solver on a SUN Sparc SOLARIS2.6 machine. Each run was limited by 8 hours. We present the results in Table 3. The first column in the table shows the relative difference between the best solution found by the FATCOP sequential solver and the known optimal solution. If the optimal solution is found, column 2 shows whether the solution is a proven optimal solution or not. Execution time in column 3 is clock elapsed time, and does not include GAMS compilation and solution report time. Tree size at the time when the program was terminated is given in column 4. The last column reports the parameter $\tilde{U}$ we set for each problem. The FATCOP sequential solver is able to find provable optimal solutions for half of the test problems.

The test problems then were solved by the FATCOP parallel Condor-PVM solver. The number of problems N generated in the initial stage was set to 20. At the beginning the master sends 40 requests for new hosts to Condor. FATCOP implements an asynchronous algorithm, hence communication may occur at any time and is unpredictable. Furthermore, the number of workers in the life cycle of a FATCOP job keeps changing so that the BB process may not follow the same path for different executions. Our experiments show that the search trees were almost never expanded in the same order for a given problem. This feature often leads FATCOP to different execution time. We ran 3 replications for each problem. For all the runs, FATCOP found provable optimal solutions for the test problems, including *fiber* that can not be solved effectively by the sequential solver. We report the average execution time, search tree size, resource utilization and resource losses in Table 4. For all runs except four, FATCOP lost some workers, but the program recovered correctly and returned correct solutions. Therefore FATCOP was tolerant to the resource retreats in our experiments. During one run for solving *fiber*, FATCOP lost many machines because it ran over the daily Computer Sciences Department reboot period.

We compare the results obtained by sequential solver on problems that it solved to optimality with those obtained by the parallel solver in Table 5. Of the problems solved to optimality by the FATCOP sequential solver, *blend2* solves quicker using the sequential code. between the sequential and parallel solver. *Stein45* and *vod3* require a few more node evaluations (but less time) by the parallel solver, while *bell5* and *misc07* solve quicker with the parallel solver. Run time for *bell5*, *stein45*, *vod3* and *misc07* is reduced by factors between $0.6 - 6.5$. Table 4 shows that FATCOP was able to collect many machines during its execution, but many factors prevent further speedup. These factors include long starvation for workers, enlarged search trees, resource failures, and inefficient communication due to PVM's nature. For *blend2*, the parallel solver even took longer than the serial code to solve the problem. We keep this problem in the test set to show that it is only beneficial to use FATCOP to solve long-running complex MIPs.

The last experiment we performed submitted a long-running MIP problem to Condor. The problem we solved *vpm1* is also from MIPLIB that has 234 constraints and 378 variables, of which 168 are integers. The FATCOP sequential solver could not find the provable optimal solution for this problem in 8 hours. We configured the parallel solver to use pure pseudocost-estimation searching and pseudocost branching. Our experiments showed that pseudocost-estimation searching strategy usually keeps a very small work pool, so the program requires small memory and takes relatively short time to write the work pool information to the disk.

The strategy also is able to find good feasible solutions quickly, so it is particularly suitable for long-running big problems. The job finished in 39.2 hours with provable optimality. It ran over two Computer Sciences Department daily reboot periods, used 21 machines on average and had 447 machines suspended during the run. To test fault tolerance of the master, we let FATCOP record the work pool information to disk every 100,000 node evaluations. We interrupted the job once (to simulate a master failure) and re-submitted the problem to Condor. FATCOP then read in the MIP problem as well as the work pool information, and started from where the work pool information was last recorded. This indicates that FATCOP is tolerant to both worker and master failures.

5 Summary

MIPs often represent critical decision problems in industry, but many of them cannot be solved sequentially on a single computer. In this paper, we provide a parallel implementation for MIPs using distributed privately owned workstations.

The solver, FATCOP, is designed in the master-slave paradigm to deal with different types of failures in a distributed ownership environment with the help of Condor, a resource management system. To harness the available computing power as much as possible, FATCOP uses a greedy strategy to acquire machines.

FATCOP has successfully solved some real life MIP problems such as the applications to video on demand system design [8] and product design [25]. It was also tested on a set of standard test problems from MIPLIB [18]. FATCOP was able to solve some problems which can not be solved by its sequential version, and achieved speedup of factors between $0.6-6.5$ for some reasonable large problems soluble by its sequential version. We leave experiments on MIP problems requiring days or months of computation as future work. Problems with extremely large search trees were excluded from our test set, since they remained insoluble in a reasonable amount of time. Solving such problems may require some specialized techniques. For example recent work that combines BB with cutting planes has been very successful [5]. Adding cutting planes at the top of node for BB algorithm can substantially reduces the size of the search tree, and the overall computation time. This cut-and-branch framework might be used in a future version of FATCOP.

6 Acknowledgement

The authors are grateful to both Miron Livny and Michael Yoder for advice and assistance in using Condor and the Condor-PVM environment. We also wish to thank Jeff Linderoth for his very insightful comments.

References

[1] M. Avriel and B. Golany. *Mathematical Programming for Industrial Engineers*. Marcel Dekker, 1996.

[2] E. Balas and C. H. Martin. Report on the session on branch and bound/implicit enumeration, in discrete optimization. *Annals of Discrete Optimization*, 5, 1979.

[3] M. Benichou and J. M. Gauthier. Experiments in mixed-integer linear programming. *Management Science*, 20(5):736–773, 1974.

[4] A. Brooke , D. Kendrick and A. Meeraus. *GAMS: A user's Guide*. The Scientific Press, South San Francisco, CA, 1988.

[5] S. Ceria, C. Cordier, H. Marchand and L. A. Wolsey. Cutting planes for integer programs with general integer variables. *Mathematical Programming*, 81(2):201-214, 1998.

[6] Condor Group, University of Wisconsin, Madison. *Condor Version 6.0 Manual*. 1998.

[7] CPLEX Inc.. *CPLEX Optimizer*.

[8] D. L. Eager, M. C. Ferris and M. K. Vernon. "Optimal Regional Caching for On-Demand Data Delivery," *to appear in Proc. 1999 Multimedia Computing and Networking (MMCN '99), San Jose, CA, January 25-27,* 1999.

[9] J. Eckstein. Parallel Branch-and-bound Algorithm for General Mixed Integer Programming on the CM-5. *SIAM J. Optimization*, 4(4):794–814, 1994.

[10] D. H. Epema, M. Livny et al. A worldwide flock of condors: load sharing among workstation clusters. *Journal on Future Generations of Computer System*,1996.

[11] J. Forrest et al. Practical solution of large scale mixed integer programming problems with UMPIRE. *Annals of Discrete Optimization*, 5, 1979.

[12] A. Geist, A. Beguelin et.al. *PVM: Parallel Virtual Machine - A user's Guide and Tutorial for Networked Parallel Computing*. The MIT Press, Cambridge, Massachusetts, 1994.

[13] B. Gendron and T. G. Crainic. Parallel Branch-and-Bound algorithms: survey and systhesis. *Operations Research*, 42(6):1042–1060, 1994.

[14] W. Gropp, E. Lusk, A. Skjellum. *Using MPI : portable parallel programming with the message-passing interface*. The MIT Press, Cambridge, Massachusetts, 1994.

[15] V. Kumar and A. Gupta. Analyzing scalability of parallel algorithms and architectures. *Tech. Report TR 91-18, Department of Computer Science, University of Minnesota*, 1991.

[16] J. Linderoth and M. W. P. Savelsbergh. A Computational Study of Search Strategies for Mixed Integer Programming. *Report LEC-97-12, Georgia Institute of Technology*, 1997.

[17] M. J. Litzkow, M. Livny et al. Condor - A hunter of idle workstations *in Proceedings of the 8th International Conference on Distributed Computing Systems, Washington, District of Columbia, IEEE Computer Society Press*, 108-111, 1988.

[18] 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*.

[19] J. L. Nazareth. *Computer Solution of Linear Programs* Oxford University Press, 1987.

[20] G. L. Nemhauser, L. Wolsey. *Integer and Combinatorial Optimization* Wiley Interscience, 1989.

[21] G. L. Nemhauser, M. W. P. Savelsbergh and G.S. Sigismondi. MINTO, a Mixed Integer Optimizer. *Oper. Res. Letters*, 15:47–58, 1994.

[22] J. Pruyne, M. Livny. Providing Resource Management Services to Parallel applications. *Proceedings of the Second Workshop on Environments and Tools for Parallel Scientific Computing, May, 1994.*

[23] E. A. Pruul and G. L. Nemhauser. Branch-and-Bound and parallel computation: a historical note. *Oper. Res. Letters*, 7:65-69, 1988.

[24] M. W. P. Savelsbergh. Preprocessing and Probing for Mixed Integer Programming Problems. *ORSA J. on Computing*, 6: 445–454, 1994.

[25] L. Shi, S. Ólafsson and Q. Chen. An optimization framework for product design. submitted to *Management Science*, 1998.

[26] R. Wunderling. Documentation of the SOPLEX library.
http://www.zib.de/Optimization/Software/Soplex/.