A DISTRIBUTED BRANCH AND BOUND METHOD FOR BOINC DESKTOP GRIDS

Bo Tian¹, Mikhail Posypkin¹
¹ Moscow State University, Moscow, Russia

GRID’2012
16-21 July, 2012
Dubna, Russia

GRID’2012 The 5th International Conference “Distributed Computing and Grid-technologies in Science and Education”
Desktop Grid For International Science Collaboration
Outline

• Introduction;
• Problem definition and method of solution;
• How to improve load balance;
• Test case – 0-1 knapsack problem;
• Test results;
• Conclusions(Future work).
Introduction

The Branch-and-Bound method (B&B) is a very efficient and well-known technique to solve combinatorial optimization problems. Many parallel B&B approaches have been proposed so far. However, distributed grid-oriented B&B implementations are not well studied. That’s why we do this study.
BOINC & SZTAKI Desktop Grid

Our project created with the help of SZTAKI Desktop Grid package which is a featured BOINC distribution.

**BOINC** (Berkeley Open Infrastructure for Network Computing) is an open source platform for Desktop Grid computing. The BOINC software consists of two parts: server software that is used to create volunteer computing projects and client software.

**Sztaki Desktop Grid:** is basically a BOINC server packed in a Debian@package to make server deployment as easy as possible.
Fault-tolerance

BOINC itself are fault-tolerance.

• **Master side:** since it can have separate scheduling and data servers with multiple servers of each type. Thus, if one of these servers is down another will guarantee the execution of BOINC tasks.

• **Client side:** BOINC has a Monitoring module. It’s easy to trace back faulty Work Units;
Load balance

A. Master send work-unit to client

B. processing time (fixed)

Task 1 not completed

Task 2 completed

Task 1 completed
Sequential Branch and Bound

The feasible solutions of BnB are organized as a search-tree, and each node is a partial solution, i.e. a part of the solution space.

Aim to find the max/min value, there are three operations:

1. Branching: split in sub-problems;
2. Bounding: compute lower/upper bounds;
Distribution Strategy

Master          Client A          Client B
1. Master do branching operation, to get a possible solution;
Distribution Strategies

1. Master do branching operation, to get a possible solution;

2. Master do bounding operation, to calculate the lower/upper bound;
Distribution Strategies

1. Master do branching operation, to get a possible solution;
2. Master do bounding operation, to calculate the lower/upper bound;
3. Master do pruning operation, eliminate bad branches.
Clients also do these three operations, but in limited period.
Distribution Strategies

Master

Client A

Client B

pruned branch

Up to maximum steps, client stop and return better result, better lower bound or all the rest of branches
Distributed Branch and Bound for BOINC

1. State chart for Master

A: start state

B: Generates the first possible solution;  
C: Branch and Bound  
F: Send branch nodes to clients;  
G: Collect results from clients;  
H: Processing  
I: Recreate branch nodes(if necessary)  
J: Send to clients  
K: end state
Distributed Branch and Bound for BOINC

2. State chart for Clients are simple

A: start state
B: Get Work Unit from master (one or server branches);
C: Computing when CPU are free;
D: Return result to master.
E: end state
0-1 knapsack problem is a “easier” NP-hard problems but still hard to solve for a variety of new test problems with larger coefficients.
3 groups of randomly generated instances

(1) Uncorrelated data instances;  
(2) Weakly correlated instances;  
(3) Strongly correlated instances.

Experimentation

There are several ways to deploy the BOINC server

a. The easiest way to set up a BOINC server is to use a BOINC server VM;

b. Can use any Unix system as a BOINC server;

c. Deploy a BOINC server on the Cloud, e.g. Amazon Elastic Computing Cloud
Experimentation

Our Experimental project is created on a single machine with Ubuntu system, this computer acts both BOINC server and BOINC client. At present we only checked how many Work Unit can be created by our program. And make sure each Work Unit can return the expected result. In future we will implement it to distributed environment and check how effectiveness our system works.

![Graph showing the number of Work Units vs number of elements in KP for different instances.](image-url)
Why don’t we use message?

The BOINC and DC-API provides limited messaging functionality between the master application and the clients.

1. Messages are not reliable in the sense that if the client is not actually running when a message is being sent to it (e.g. because it is queued by the backend grid infrastructure), then the message may be silently dropped.

2. The ordering of messages is not necessarily maintained.

3. Messages are delivered asynchronously. There is no limit for the time elapsed before a message is actually delivered.
Future works

1. Go large scale;
2. Try hybrid infrastructure (DG+cloud computing, DG+supercomputing);
3. Implement “message”;
4. Go production.
Thank you for your attention!