EFFICIENT IMPLEMENTATION OF BRANCH-AND-BOUND ON DESKTOP GRIDS

Bo Tian[1], Mikhail Posypkin[1]
[1] Moscow State University, Moscow, Russia

GRID'2014
1-5th July, 2014
Dubna, Russia

GRID'2014 The 6th International Conference
"Distributed Computing and Grid-technologies in Science and Education"

Desktop Grid For International Science Collaboration

Outline

- **Introduction;**
- Problem definition and method in solution;
- How to improve load balance;
- Test case – 0-1 knapsack problems;
- 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.

Outline

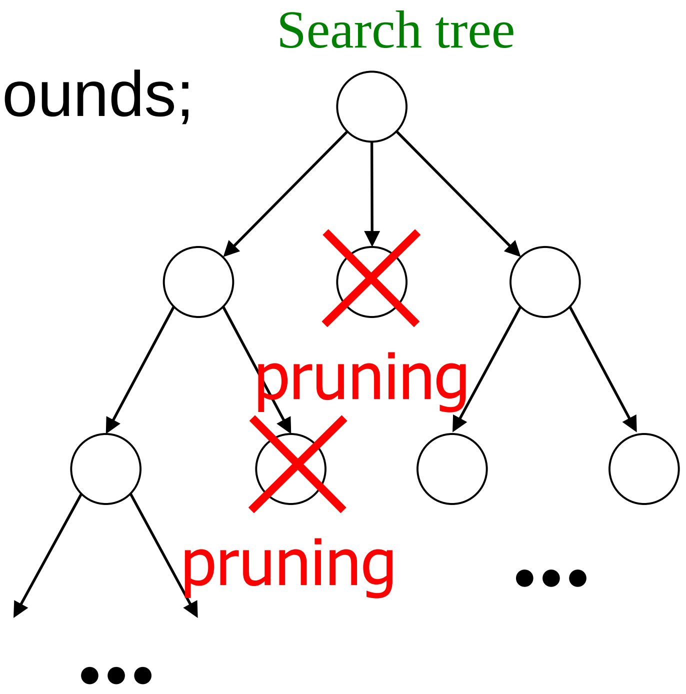
- Introduction;
- **Problem definition and method in solution;**
- How to improve load balance;
- Test case – 0-1 knapsack problems;
- Conclusions(Future work).

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;
3. Pruning: eliminate bad branches.

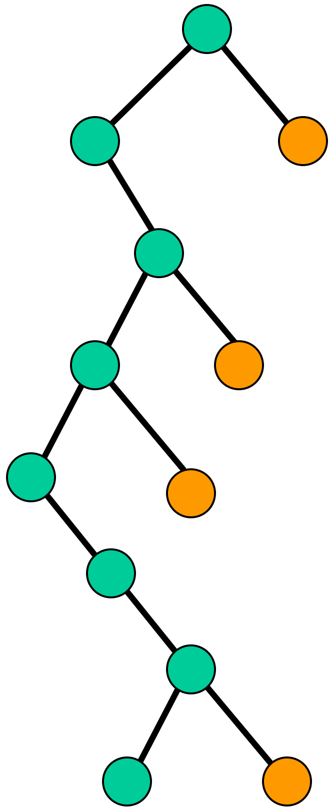


General Distribution Strategy

Master Client A Client B

General Distribution Strategy

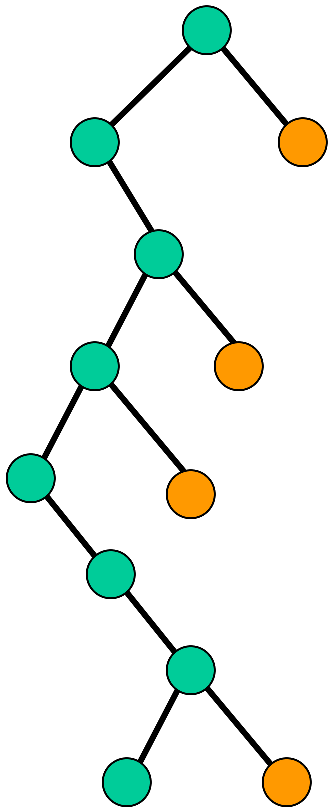
Master Client A Client B



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

General Distribution Strategy

Master Client A Client B

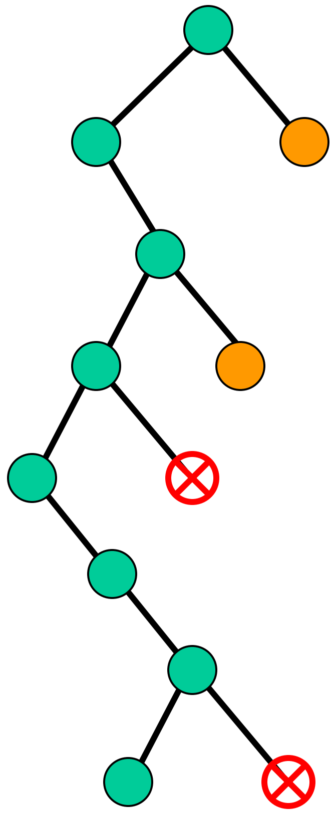


1. Master do branching operation, to get a possible solution;

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

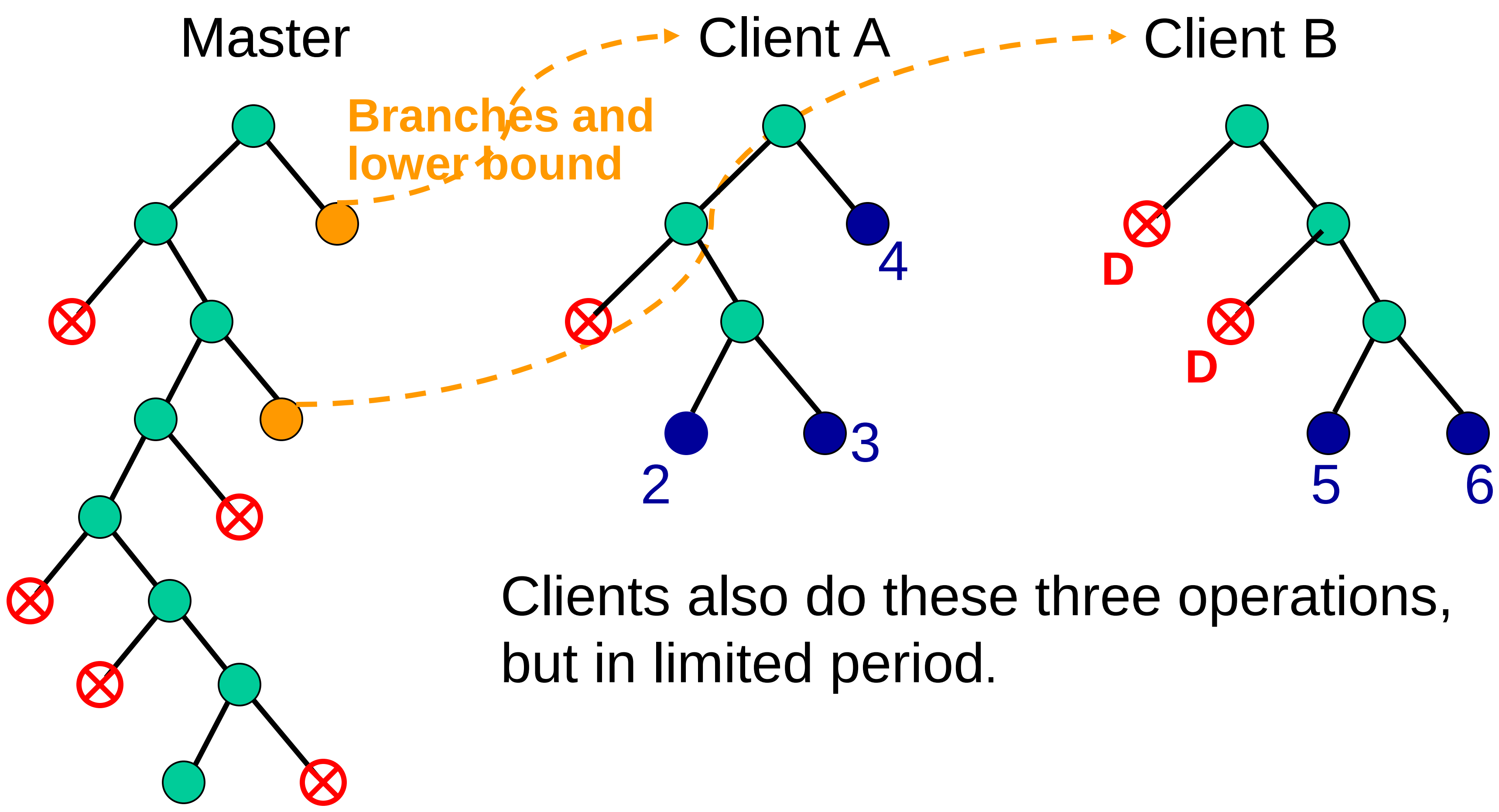
General Distribution Strategy

Master | Client A | Client B



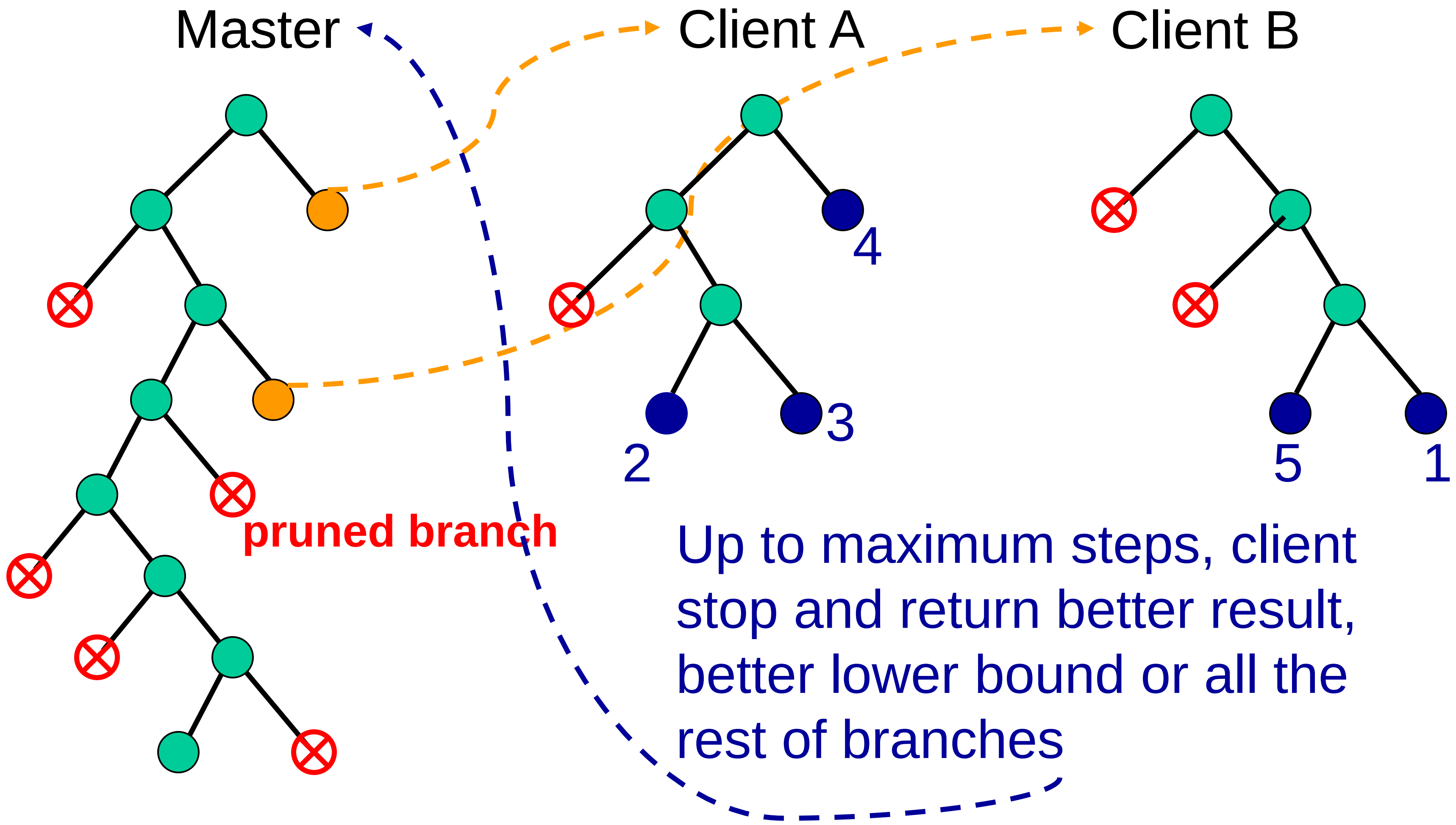
1. Master do branching operation, to get a possible solution;
2. Master do bounding operation, to calculate the lower/upper bound;
3. Master do pruning operation, eliminate bad branches.

General Distribution Strategy



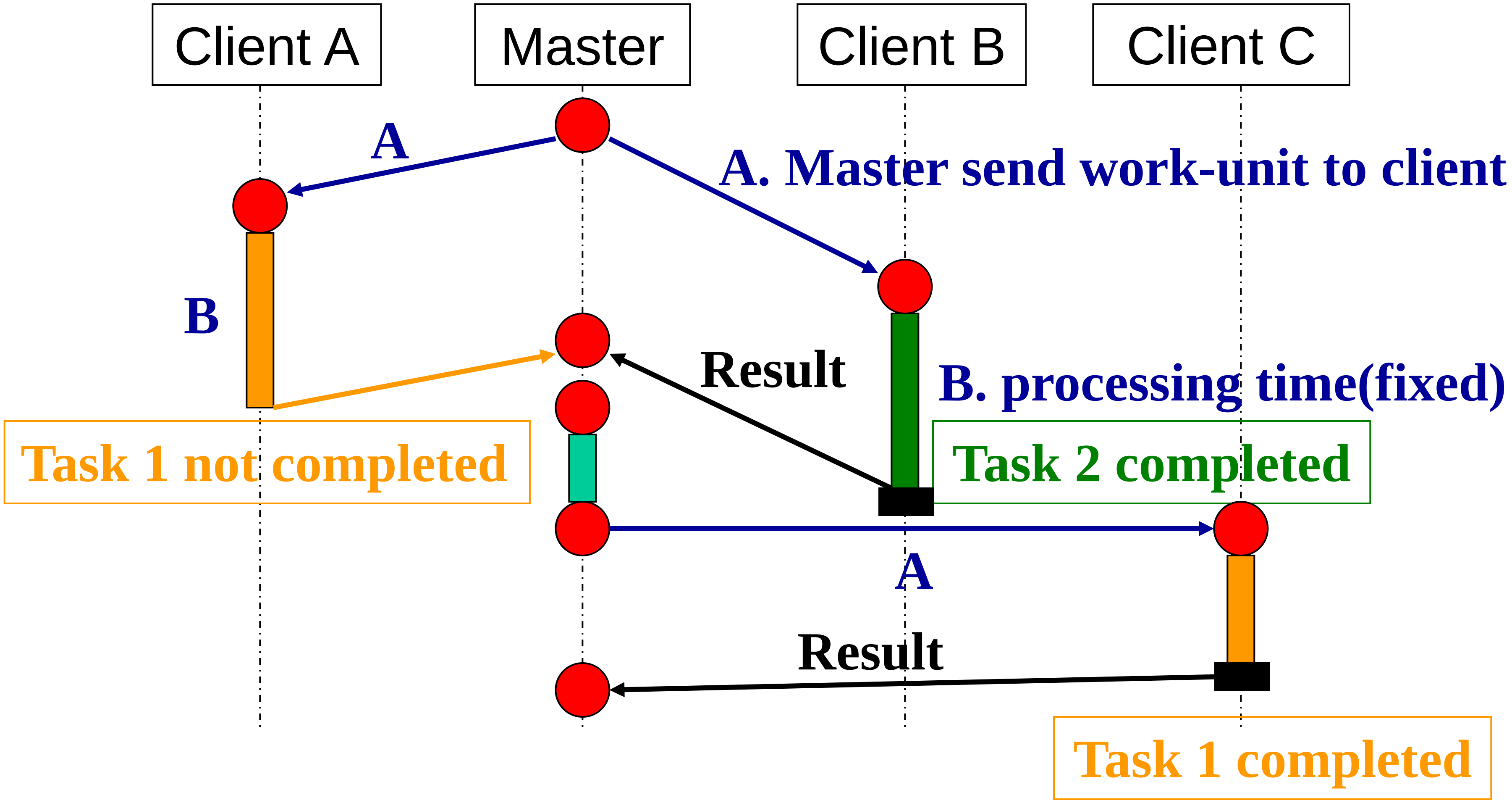
Clients also do these three operations, but in limited period.

General Distribution Strategy



Up to maximum steps, client stop and return better result, better lower bound or all the rest of branches

General Load Balance Strategy



Outline

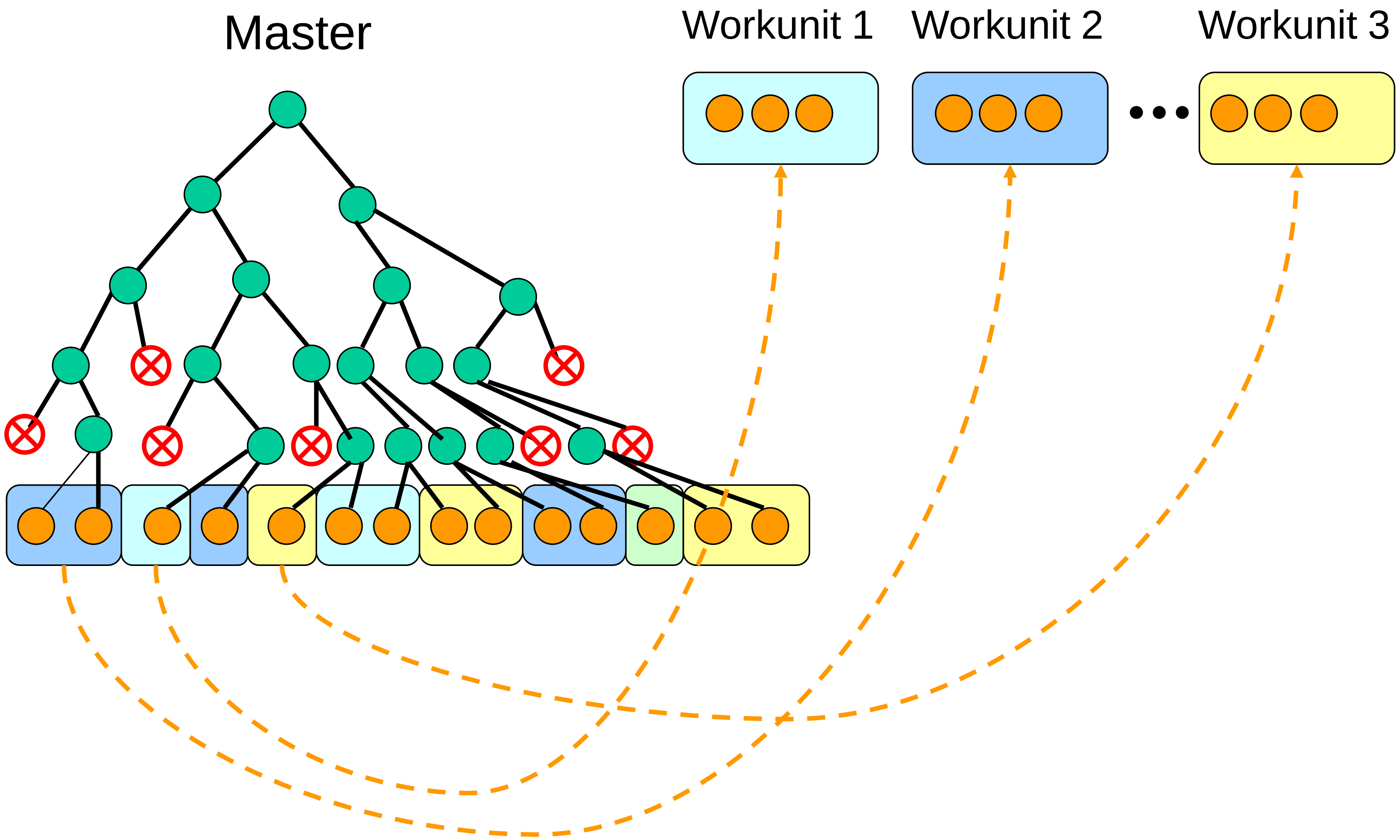
- Introduction;
- Problem definition and method in solution;
- **How we improve load balance;**
- Test case – 0-1 knapsack problems;
- Conclusions(Future work).

Why don’t we use message?

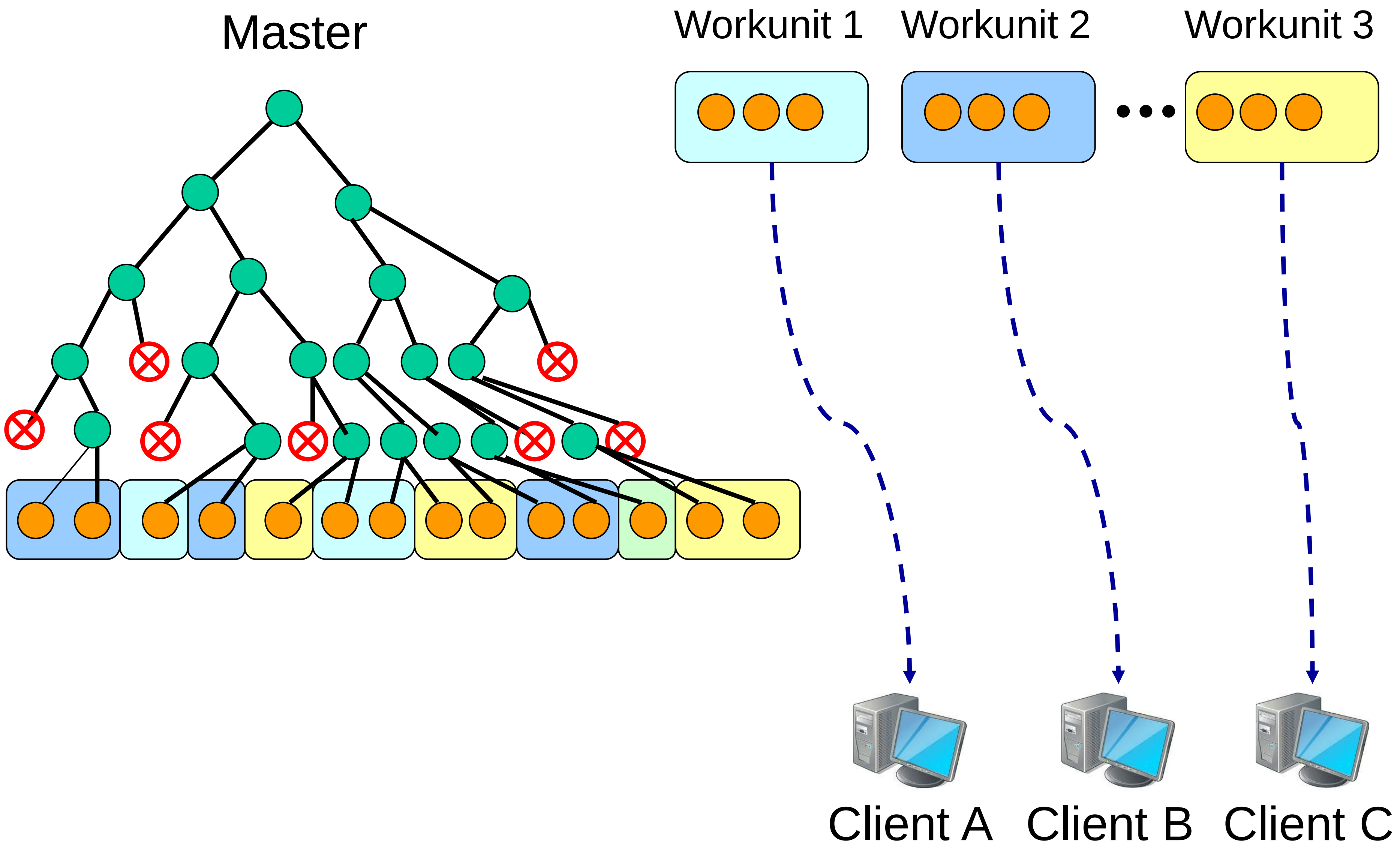
The BOINC and DC-API provides limited messaging functionality between the master application and the clients.

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

Our New Distribution Strategy



Our New Distribution Strategy



Our Load Balance Strategy

A. Dense strategy: Dense strategy packages the physically close sub-problems from the search tree into W workunits.

$$w_i = \left[s_{\frac{(i-1)S}{W}+1}, s_{\frac{(i-1)S}{W}+2}, \ldots, s_{\frac{iS}{W}} \right] \tag{2}$$

where i=1,2,...,W

B. Sparse strategy: Sparse strategy equidistantly picks subproblems then packages them into W workunits.

$$w_i = \left[s_{\frac{(i-1)S}{W}+i}, s_{\frac{iS}{W}+i}, \ldots, s_{\frac{(i-2+\frac{S}{W})S}{W}+i} \right] \tag{3}$$

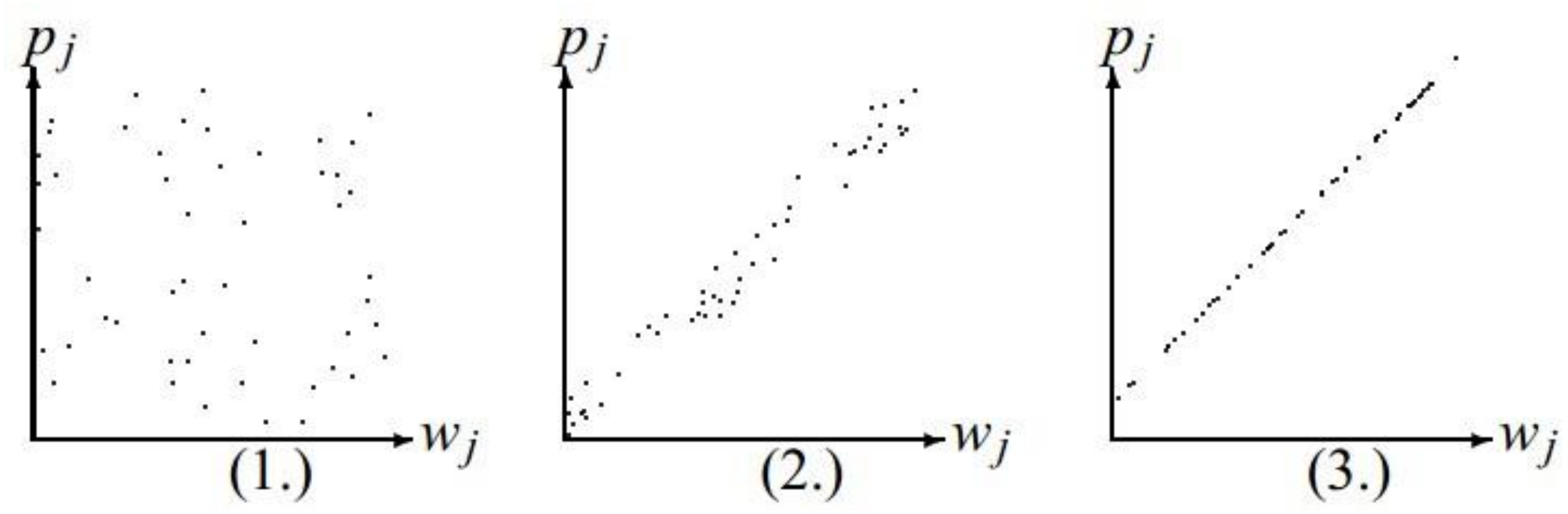
where i=1,2,...,W

C. Random strategy: Random strategy we use FisherYates shuffle [14] to generate a random permutation of all subproblems set, in plain terms, for randomly shuffling the subproblems. Then package them into W workunits. The pseudo code

Outline

- Introduction;
- Problem definition and method in solution;
- How we improve load balance;
- **Test case – 0-1 knapsack problems;**
- Conclusions(Future work).

Our Load Balance Strategy



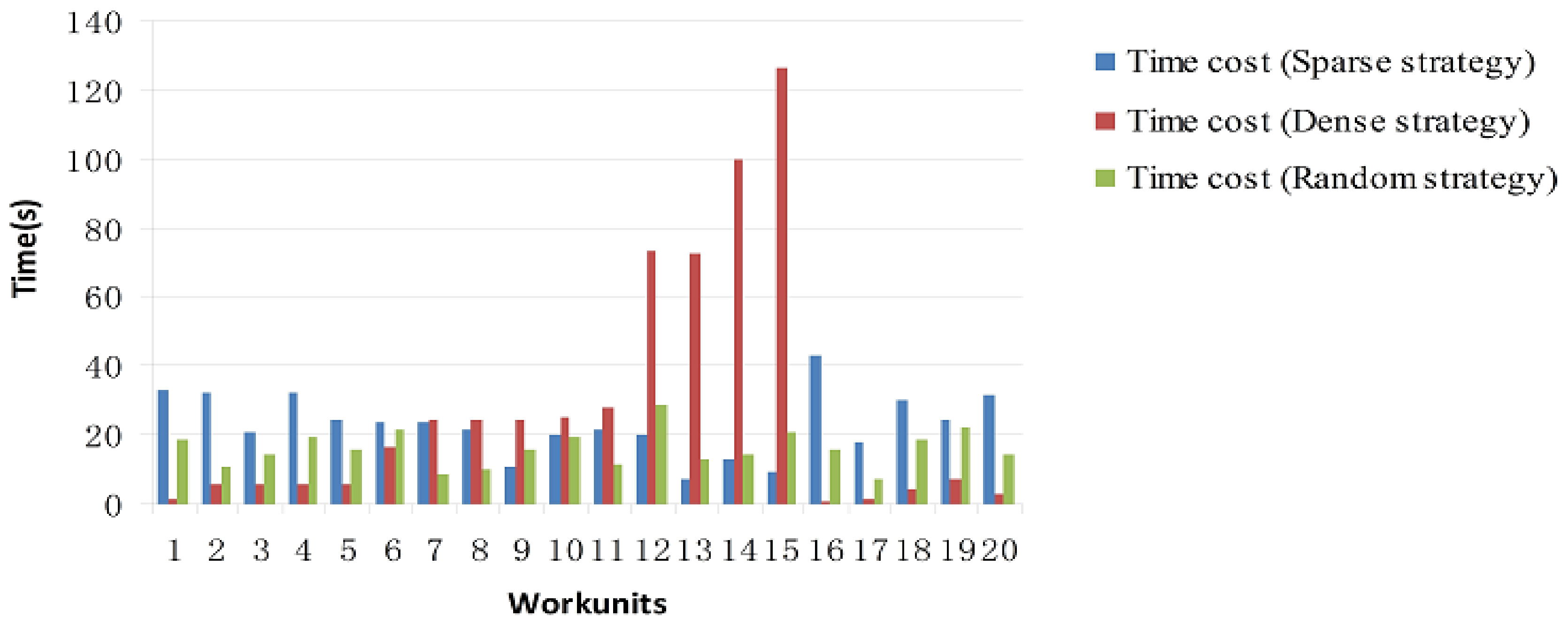
(1) Uncorrelated data instances;

(2) Weakly correlated instances;

(3) Strongly correlated instances.

Experimentation

We tested BNBTEST@HOME on a small cluster with 15 computers, each with 2-4GB of RAM and 2-8 cores processor, running different operation systems, primarily GNU/Linux and Microsoft Windows series.



Outline

- Introduction;
- Problem definition and method in solution;
- How we improve load balance;
- Test case – 0-1 knapsack problems;
- **Conclusions(Future work).**

Future works

BNBTEST has been proven as a good distributed branch and bound solver. Future work will focus on more efficient packaging and distribution strategy, and make the BNBTEST to be a modular middleware, with the user interface for user. Also we plan to increase our volunteer grid for solve more complex and practical optimization problem.

Thank you for your attention!