A Development and Deployment Framework for Distributed Branch & Bound

The branch-and-bound algorithm proves that a solution can be found without going through all feasible solutions because it can be found from the problem description itself. This could though be computationally infeasible because the possible solutions grow exponentially with the problem size. Of all the feasible solutions generated, we can create a search tree, where each tree node has a partial feasible solution. This is called *branching*. The set of feasible solutions the nodes represent are an extension of its partial solution. If a node consists of feasible solutions that are more costly than the feasible solutions already found, we *prune* this node, meaning that we do not investigate its feasible solutions further.

The paper uses a framework which supports adaptive parallelism and is designed to be distributedly used. Adaptive parallelism means that the number of compute servers can grow - they are not fixed.

When using branch and bound computing in this setting, the problem is divided into tasks which is computed on different compute servers, a process which requires task balancing. The tasks need to be created during the execution, the compute servers must propagate the best cost to the other servers, and a bottleneck of communication can not happen.

There has been a lot of related work to branch and bound. They found speedup anomalies in parallel branch and bound. A randomized Local Best First-Search algorithm was introduced for parallelizing sequential branch and bound, but ignored communication overhead and maintenance of the local data structure. When it comes to distributed branch and bound related work, they had problems with fault tolerance when the amount of processors increased..

The deployment architecture in this article use an ongoing project called JICOS, which is a Java-centric framework which supports parallel computing, stores partial results, is hardware/OS independent and also scales from a local area network to Internet. By using many hosts, or compute servers, who can join the execution, they reduce the completion time. JICOS must also tolerate faults and hide communication latency. JICOS consists of three service classes of components; the **Hosting Service Provider** is the interaction point with the clients, and acts as an agent for the whole network by propagating login/logout to the servers/hosts. The resource consumption information for the server is provided and aggregated to the clients by the HSP. The **Task server** consists of task objects which encapsulate the computations. It also holds the uncalculated spawned tasks, and it manages the load balance among tasks. The **Hosts** request tasks, execute them and returns the results, and finds new tasks to execute. A host works as a compute server.

In order to tolerate faults, it can heal itself by leasing proxy objects. If a host has an expired lease, the lease manager reassigns the hosts task and deletes it from the tasks server. Thus recomputation remains minimized.

When it comes to improving the performance, JICOS makes use of some application-controlled directives, task caching, task prefetching and task server computation. Task caching caches the tasks on its hosts, and the hosts need to ask the

TaskServer for another task. Task prefetching hides the communication latency, both implicit and explicit. Task server computation is done when a tasks' encapsulated computation is a bit more complex, and it is therefore faster that the server computes it rather than the host.

The computational model consists of a Directed Acyclic Graph (DAG), where nodes represent tasks and the arc from one node to another represents the input and output. Each task holds an immutable object as input and a shared object it has access to in the environment. The shared objects enables the network. These shared objects has the least cost known solution at all times, making it possible to prune.

The environment reduces task information needed which reduces the time to unmarshal and marshal objects by setting the environment input to the problem instance.

The framework for branch and bound which JICOS uses is based on that the problem needs to be a minimization problem and that the costs is represented as integers. To use this framework, the Solution interface must be implemented, and represents nodes. The Solution objects consist of the partial feasible solution. The solution interface consists of several methods which enable the branch and bound algorithm to work properly. The MinSolution methods always hold the solution object with the least cost, making pruning possible.

The authors performed several experiments with the framework and found out that fault tolerance was highly efficient when using faulty compute servers. This was proven when faults happened, and when they did not. JICOS' efficiency as a distributed system was confirmed by having little overhead of the task servers.