Summary of “A Development and Deployment Framework for Distributed Branch & Bound”

By Torgeir Lien and Oeyvind Rein

The concept of Branch & Bound is described using TSP as an example. The simplest solution to TSP is to simply explore every branch that is formed by the set of nodes but this approach is very computationally intensive since the complexity of TSP scales rapidly with the number of nodes. By using the length of best known path so far and some knowledge about graphs, Held-Karp in this case, it is possible to detect that the path one is currently exploring cannot possibly be a feasible solution. The path can thus be discarded along with all it's sub-branches something which saves the computer from a lot of useless work. A good way to solve problems like this is required.

Related work concerning B&Bis then discussed. Important topics include the “invention” of B&B, the computational complexity when running B&B in parallel, distribution, fault tolerance and various frameworks.

The framework that is presented is based on Java and called Jicos. It is designed for adaptive distributed parallelism which makes it possible to increase and decrease the number of computers in the system dynamically while requiring certain things of the compute servers. The framework is an ongoing project and the goal is that it should reduce completion time of tasks and virtualize compute cycles, store partial results, have fault tolerance, be partially self-organizing, be hard/soft-ware independent and should scale from LAN to the Internet. The system has three component types, HSP, server and host. The framework tolerates faulty hosts by using leasing, and it supports self-healing.

Several concepts for optimizing performance is incorporated by Jicos. Tasks will cache sub-tasks in the local computer to reduce the need for sending tasks to and from the server. Task pre-fetching is used to gather tasks before the computer starves for work to do. Task server computation will make sure that small tasks are computed in the server instead of having to travel over long latency links.

The computational model of the framework is a DAG which represents tasks, inputs and outputs. Tasks will be divided into smaller tasks until it is deemed 'atomic' and one computer will execute it. The environment sets input to the problem instance to reduce the amount information needed to describe a task. It uses shared objects to share information about the best cost found between computers.

Jicos makes two assumptions for the classes used; the B&B problem is formulated as a minimization problem and the cost can be represented as an int. A list of problem-specific classes that the programmer must provide is given, it includes classes like reduce, isComplete and minSolution.

Experimental results solving TSP with Jicos is then given. Tests were done on one and several computers with in total as many as 120 cores. Fault tolerance and overhead was also tested. The speedup of adding processors exceeds 94 % for all test cases. Better than perfect speed up was achieved for test between 4 to 32 processors. This is explained with the fact that object placement in memory hierarchy can have impacts that are greater than the added overhead. Maximum parallelism is calculated to be P = 1857 processors which will bring computation time down to 37 seconds. The measured overhead of communication when testing fault tolerance is less than 10 % for all tests between 4 and 30 processors.

In conclusion the framework should make programming distributed B&B programs easier and less error prone. It provides good performance and speedup, tolerates faulty computers while keeping communication and task server overhead small. The source code is cleanly designed, easily changed and available for free to any one who wants to use it.