A development and deployment framework for distributed branch & bound

Branch and bound is a technique applied to a combinatorial optimization problem in order to reduce the number of feasible solutions one has to calculate. The number of feasible solutions for a problem typically grows exponentially as the input size increases, thus reducing the number of computations is crucial. Using branch and bound in solving TSP, the space of feasible solutions can be partitioned into a search tree. When traversing this tree, we may come across a node that represents feasible solutions, all of which have a higher cost than the current best solution, thus we can prune the node from the tree. To keep track of the current best solution in a distributed setting, we need to propagate the current best solution to all compute servers.

The term branch and bound first appeared in 1963 to describe an enumerative procedure for solving TSP instances by Little, et al. Distributed branch and bound computation became an interesting topic for many studies, and Yahfoufi and Dowaji presented the first fault-tolerant branch and bound algorithm. Later, PUBB (Parallel Utility for Branch-and-Bound) by Shimano et al, first introduced the notion of an LCU (Logical Computing Unit). This paper presents JICOS, a Java-centric network computing service which supports high-performance parallel computing. JICOS applies branch and bound to solve computationally difficult problems like TSP.

JICOS is designed to support scalable, adaptively parallel computation, tolerate basic faults and hide communication latencies. It consists of 3 service component classes:

1. Hosting Service Provider (HSP): The component that JICOS clients interacts with (client log in, submit tasks, requests results, and logs out). The HSP acts as an agent for the entire network of service components, and it manages the network of task servers. It also propagates logins and logouts of a client to all their task servers which then propagates it to all their hosts.
2. Task Server: A store of Task objects that also balances the load of ready tasks. It also handles fault-tolerance by reassigning a host's tasks if a host fails.
3. Host: A compute server joins a task server, requests and executes tasks.

The JICOS API is application-controlled in the sense of having directives for improving performance by reducing communication latency, and it supports task caching, task prefetching, and task server computation. Computations are modeled as directed acyclic graphs (DAG), in which nodes represents tasks and an edge between two nodes is the input of a task provided by the a task. The tasks have access to an immutable input object and a mutable shared object. In the search tree, nodes corresponds to tasks, which are decomposed to subtasks until we get to the threshold of an atomic task (a task being small enough to be computed by a single computer). The branch and bound framework of JICOS is based on the assumptions that the input problem is a minimization problem, and that the cost can be represented as an integer.

While running JICOS on a Linux cluster when applied to a euclidean TSP-problem with 200 cities, the original problem instance decomposed to 61,295 Tasks, with an average execution time of 2.05s. T_∞ (critical path) of this problem was 37 seconds, while T_1 depended upon the number of processors (from over 9 hours with 2 processors to just 11 minutes with 120 processors). The results show superlinear speedup for 4, 8, 16, and 32 processors. While measuring the overhead of running the task server on a machine shared with a compute server, the average was 3115.1 seconds for having a dedicated task server, and 3114.8 for the shared case, both of which represent a speedup of 99.7%, thus it appears beneficial to place a compute server on every available computer in a JICOS system without dedicating machines to task servers.