Anticipated Distributed Task Scheduling for Grid Environments

A Summary

Morgan Ericsson

Växjö universitet

Many scientific algorithms have an inherent structure than can expressed as a series of interacting modules. This inherent structure can be represent using a (task) graph, with modules (tasks) as nodes, and control and data-dependencies as edges in the graph. The tasks of a task graph can either be single processor or multiprocessors (M-Tasks) tasks. Each M-Task displays two levels of parallelism: a M-Task can be executed in parallel on several processors and several M-Tasks that are not dependent on each other can be executed concurrently. In order to execute a M-Task it has to be scheduled to a set of processors.

In a Grid computing environment there are often large heterogeneous computing resources connected via networks. These computing resources are shared between several users. Compared to other parallel computing resources, this means that the optimal scheduling can change due to "unexpected" load changes. So, in order for a scheduling algorithm to counter these changes, it could be dynamic, i.e., moving tasks between computing elements in order to satisfy the application constraints, i.e., maximal throughput etc.

The scheduling of an M-Task is done in two steps. First, the graph is partitioned into a sequence of layers, $L_i, i = 1, 2, \ldots, N$, where each layer should contain as many concurrent tasks as possible. These layers are then scheduled to be executed sequentially. Once the partitioning into layers are done, the tasks of each layer is scheduled, either to execute in parallel or in sequence, and whether to execute it locally or remotely. The decision where to place layer L_{i+1} is taken once layer L_i is placed. This is so that the cost of migrating a task M, $C(M)$ can be hidden by the execution of the previous layer. When trying to place layer L_{i+1}, all the tasks of the layer are divided into two sets, those migratable and those not migratable. A task is considered non-migratable if the cost of migrating it can't be hidden by the execution of the previous layer of if it depends on input data not yet available.

In order to migrate a task we need to find a remote server S_j such that the total accumulated execution time of L_{i+1} is lower on S_j than it would be on the current server. If such a server can be found, we select a migratable task M from layer L_{i+1} such that is has the longest execution time and we then check if the total execution time of server S_j including task M is still lower than that of the local server. If this is the case then the task is migrated.

In order to show the sub-optimality bounds of the algorithm, we first define a ration α

$$T(M_x, p_1) = \alpha \times (T_{i+1}(S_1) - T(M_x, p_1))$$

where M_x is the task with the smallest execution time, $T_{i+1}(S_1)$ is the total execution time of layer L_{i+1} on server S_1, and $T(M_x, p_1)$ the execution time of task M_x on processor p_1.

If we assume that $0 \leq \alpha < 1$ and $1 - \alpha \times \frac{p_1}{p_n} > 0$ we have the following sub-optimality bound.

$$T_{i+1} \leq \frac{1 + \alpha}{1 - \alpha \times \frac{p_1}{p_n}} T_{OPT}$$

For a large number of tasks α is close to zero, which makes the algorithm close to optimal.