Summary of “How to Build a ComputeFarm”

By Torgeir Lien and Oeyvind Rein

ComputeFarm is an open source Java framework to assist the programming and running of parallel software. Parallel computing have many applications such as SETI@home, NPC problems which require some sort of brute-force solution and many other large tasks which can be composed into smaller sub tasks. ComputeFarm runs on Jini which handles the network and fault tolerance.

ComputeFarm uses the Master-Worker pattern where the master keeps track of tasks to be done and the workers ask the master for work when they are ready for it. The workers then send the results of the task back to the master or the “ComputeSpace”. The client in this system has a job which it divides into sub tasks which is then dispatched to the ComputeSpace. The ComputeSpace will wait until all tasks are done and then return all the sub-results back to the client. The client doesn't need to think about who actually does the work.

The paper now takes on a more practical view and provides an example on how to split tasks and how to use the framework to simplify the job of running them in parallel. JobRunners from the JobRunnerFactory takes care of the clients job. The client calls run on job and is then blocked until a result is ready. The papers then describes briefly how fault-tolerance is done in the setting and that tasks must be serialize-able to be used with the Java RMI in order to distribute the tasks on the network.

The text then explains how code mobility can be achieved by using dynamic code downloading or more specifically the code base utility provided by the Java RMI framework. This means that Java classes will be made available for download to any of the workers that may need it. That is the worker does not need to know anything about the task or how to perform it until it is actually asked to perform the task. This is handy and make workers a whole lot more general. Since there may be some problems with setting up a separate HTTP server just for a code base the paper suggests using the embedded HTTP server called ClassServer which ComputeFarm provides. Another way is to specify a ClasspathServer which is easy to set up but have some drawbacks like the fact that every class will need a separate HTTP call which is detrimental to performance and in addition the class has no security controls.

Some ways to practically run the example program is provided. The authors recommend using Blitz JavaSpaces since it is open source. Details on what packages to download and how to install and run them are then described in addition to specifying how to set up permissions correctly.

The problem of fault-tolerance is then discussed while giving details on how to deal with the problems. Jini deals with most of the problems by throwing rmi.RemoteExceptions which the programmer must deal with. The communication of tasks between client, space and workers is transaction-based which means that the tasks will be rolled back after a failure and then for example given to another worker. Cancellation of jobs is also possible. The different kinds of exceptions that can be thrown and how to deal with them are then discussed. The three exceptions are “ComputeSpaceException”, “CannotWriteTaskException” and “CannotTakeResultException”.

The conclusion is that there are large gains to be had by using a distributed system for large tasks. One should consider using the ComputeFarm framework for parallel tasks so that they can be easily distributed on a network.