# How To Build a ComputeFarm

This article is about an open source Java framework for parallell computing called ComputeFarm. The framework makes use of the “Replicated-Worker Pattern”. Where they use Spaces for holding tasks and results as objects. In this case, the lifecycle of a worker is; wait for a task, execute a task, put the results back into the Space (ComputeSpace), and then do it again. There are several identical workers, who contributes with the resources it can. A job is created by the clients who also specify how the job should be divided into tasks, which again is available in the ComputeSpace through the Job. By the time a task has executed, the result is read by the Job in the ComputeSpace. The workers do not know about the other workers, and the processes are executed concurrently.

An example usage is the calculation of the sum of the first n squares. We can consider multiplication as an expensive operation in this case. The job is the calculcation of the first n squares, and this job could be divided into smaller tasks, a subset of the job. Each task could be the multiplication of the first n-1 squares, and in the end the solution of the different tasks gets added and we have derived the answer. In the ComputerFarm framework, the clients get a jobRunner from the jobFactory. The Job interface has two methods, one for generating tasks and one for collecting results. In the generate task method a new task gets created and each task gets written to the ComputeSpace. In the collect results methods, when a task is finished it gets added to the results. In this case, each multiplication gets added to the sum. This happends in the SquaresJob class. The SquaresTask class is simple, it only needs a execute method from the Task interface.

You can run ComputeFarm in a single JVM or multiple JVMs. When running on multiple JVMs, the task implementation must be downloaded to each JVM. The way Java RMI has solved this, is to provide a codebase URL from where you can download the classes. The class files get packed up in a *-dl.jar, making it possible to download them from a server. ComputerFarm uses ClassServer, which is a lightweight embedded server. This makes it possible to run both the server and the ComputeFarm client in the same JVM. This decreases the possibilities for configuration errors and complexity issues of running separate HTTP daemons. You can also use ClassPathServer, which is a server which runs classes directly from the classpath if you want to avoid making a .jar file.

There are different errors that may occur when dealing with parallell computing compared to single-processing. Jini has provided a framework that helps the implementor deal with these errors, as opposed to solving them. The article talks about three different kinds of exceptions that can arise when dealing with a remote ComputeSpace, and how to solve them. ComputeSpaceException is an unchecked exception that happens when a task is either written or taken from the space normally caused by limited resources, and does often not need to be handled. CannotWriteTaskException and CannotTakeResultException are both checked exceptions, whereas the first one happens during write operations, and the latter during take operations. When one of these exceptions gets throwed it is a transient communication problem with the space and it guarantees the state of the space. No take() or write() methods can change the space. This could though lead to the client blocking indefinetly the call to take(), if a call to write() has caused a cannotWriteTaskException. This could be fixed by a retry strategy, implementing the cancel() method in the Job class. Another exception that could occur is CancelledException which occurs when the cancel() method gets called. This exception can be fixed by cleaning up your code.