How to Build a ComputeFarm

What is ComputeFarm? ComputeFarm is an open-source Java framework that is used for parallel computing. The advantage of doing parallel computing is that it allows for programs to run faster, and it is done by splitting the programs to run on multiple processors simultaneously. ComputeFarm runs on top of Jini, a network architecture for construction of distributed systems.

The Replicated-Worker Pattern is a thread pool pattern, in which there is a master process that creates a set of tasks that need to be run. The workers then chooses a task from the set of tasks to run, and when the computation is done, the result is sent to the master. In ComputeFarm, there is a ComputeSpace which holds Task-objects and result-objects. The ComputeSpace is a channel for passing messages between the master and the workers, and there are usually many workers and they are identical. This pattern provides load balancing. The lifecycle of a worker consists of waiting to get a task from the ComputeSpace, executing it, put it back in the ComputeSpace, and then go back to waiting for a task. The master process (client) in ComputeFarm thinks in terms of completing a job, and it's lifecycle consists of creating a Job and specifying the tasks it consists of, and then write each task to the ComputeSpace, and as each Task is returned as a result by the workers and combined into the overall result, the Job is done. All of these processes run concurrently, and the master process may be splitting a Job into Tasks, as the results of earlier Tasks are returned, and this is because the master views the ComputeSpace as a computing resource (it doesn't know about the workers).

An example where ComputeFarm could be applied, is calculating the sum of the first *n* squares. Doing this in a simple for-loop with multiplication at each iteration is more expensive than addition. With applying concepts of parallel computing to this example, we divide the problem into smaller problems (Tasks), and then combine the sub-results to the final result. For a client to run a Job in ComputeFarm, it gets a JobRunner from a JobRunnerFactory. The Job interface specifies two methods: generateTasks and collectResults. To run this example, we start out by calling JobRunnerFactory.newInstance() which createa JavaSpacesJobRunnerFactor, and a ClasspathServer is started to allow for remote access to download the class files.

While using multiple JVMs, there must be some way for the implementations of the task to be downloaded to all of the JVMs. In ComputeFarm, the workers can be generic by dynamic code downloading, using Java RMI. This essentially means that by using multiple JVMs, there must exist a codebase where the remote JVMs can access the Java classes made available to them. ComputeFarm provides an embeddable server called ClassServer that allows you to run the server in the same JVM as the client, which is removes some of the complexity of having a separate HTTP daemon.

Fault Tolerance is an important thing to keep in mind while dealing with distributed computing. The programmer as to deal with issues regarding fault tolerance, he/she must use the provided mechanisms. The order of which services are started is not important in ComputeFarm. The system would still work if one were to first launch the client, then the workers and then JavaSpace. In case of a worker crashing, the Job computation will continue, and the worker's current Task will be rolled back and left in the ComputeSpace for another worker to execute. There are three main possible exceptions that may arise dealing with a remote ComputeSpace:

- ComputeSpaceException - Unchecked exception error that may occur while writing or taking from the space.
- CannotWriteTaskException and CannotTakeResultException - checked exceptions, the first one is thrown during writing and the second during taking. They occur if there is a short-term problem while communicating with the space.
- CancelledException - a checked exception that is thrown when cancel() is called on the JobRunner.

Conclusion: Rewriting a program as a parallel program may be very beneficial if the problems being solved can be divided into smaller subtasks and the overall problems are computationally expensive. ComputeFarm can then be used to distribute the processing across a network of JVMs.