Israel JBoss User Group

Session 07 / 6.12.2007

Grid Computing With JBoss Using GridGain

By : Yanai Franchi, Chief Architect, Tikal

Agenda

- GridGain overvirew
- Anatomy of GridGain
- Hands On With GridGain
- Jboss Meets GridGain
- Demo

What is SETI ?

- SETI (Search for Extraterrestrial Intelligence) is a scientific area whose goal is to detect intelligent life outside Earth.
- One approach, known as ***radio SETI***, uses radio telescopes to listen for narrow-bandwidth radio signals from space.

Scaling SETI

- More computing power enables searches to cover greater frequency ranges with more sensitivity.
- Radio SETI, has an insatiable appetite for computing power.
- Previous radio SETI projects have used special-purpose supercomputers, located at the telescope, to do the bulk of the data analysis.
- Now radio SETI using a virtual supercomputer composed of large numbers of Internet-connected computers - SETI@home project to explore this idea.

Sounds Familiar ?

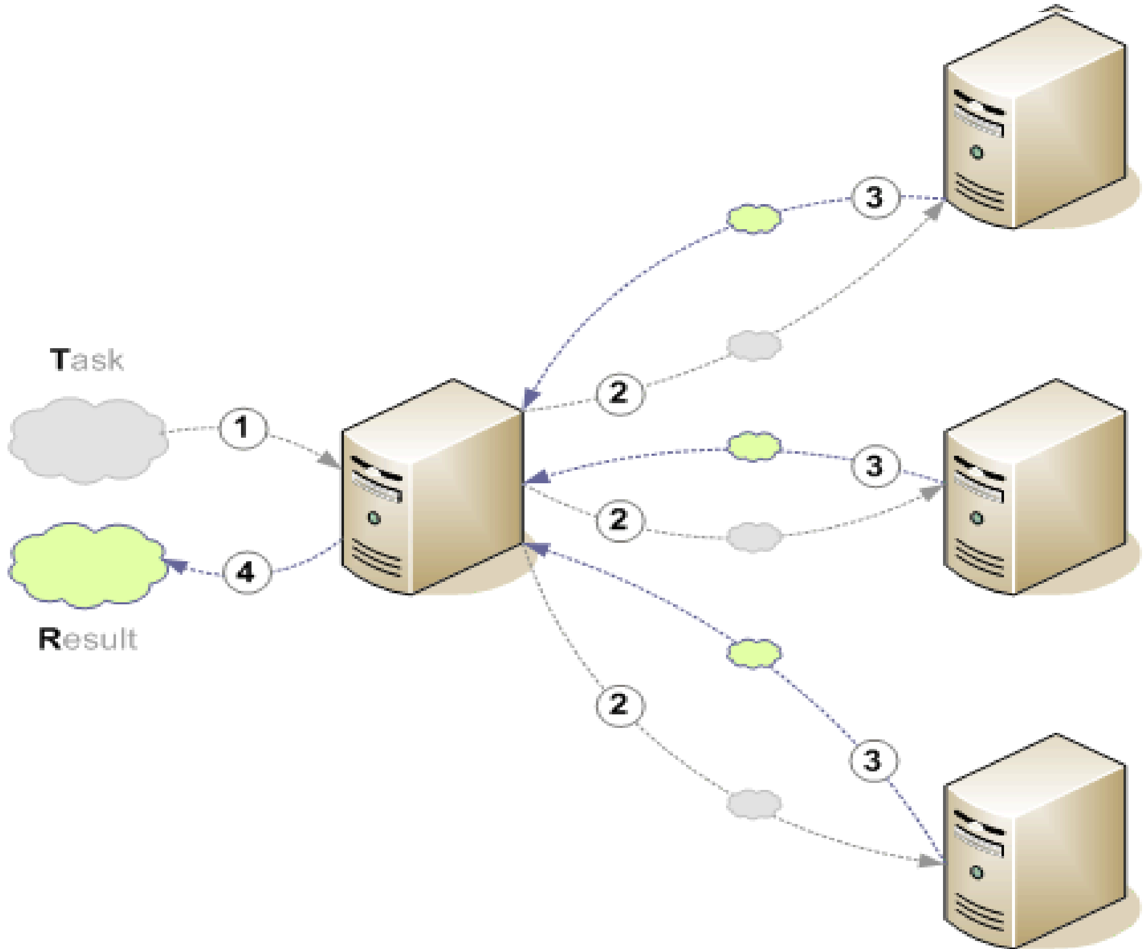
- Your business has ***a well designed and implemented*** computerized process that takes ~30 seconds to complete:
  - logs analyses, archive search, incoming transaction file processing, compressing/decompressing, etc.
- Now, you want to web-enable this process.
  - You need to shrink the timing to roughly 5 seconds
- ***How do you make something that was already well designed to run 6 times faster?***

GridGain Overview

- An innovative computational grid, combines intuitive AOP-based technology with state of the art grid computing features, create by “GridGain Systems”.
- First GA wad released in July, 4000 downloads, more than 400 active users in 4 months.
- Allows you to parallelize the execution of the piece of code onto a set of computing resources:
  - laptop, desktop computer, workstation, rack-server, mainframe or any other computing device with Java 5 or higher compatible JVM available.

Map/Reduce

- Split and aggregate (a.k.a Map/Reduce) design allows to parallelize a process of task execution, gaining performance and/or scalability



Difference From Master/Worker

- On Master/Worker a master sends task to one or more workers, workers execute the task and send results back to master. The task is being executed ***on the worker***, on the request from the master.
- With computational grid the key is the ability to ***SPLIT*** the task into logical sub-tasks, and then execute them in parallel and ***AGGREGATE*** the results back.
- You can "stretch" Master/Worker to include splitting as well, but there's a lot that goes into design to properly support split & aggregate logic.

GridGain Advantages

- Free of cost and open source(LGPL) written in Java.
- Enables AOP-based/“task oriented” grid enabling.
- “Best of Breed” Grid computing features:
  - topology management, failover, Intelegent map/reduce, pluggable deployment, checkpoint, communication, event, discovery SPIs, automatic peer-to-peer deployment, etc.
- Out-Of-The-Box integration with Spring, JBoss, AspectJ, WebLogic, WebSphere, GigaSpaces, Coherence, Mule, JUnit ...

Grid Computing Made Simple

- GridGain provides a great programing model
- Enables agile and simple development:
  - IoC-based configuration via Spring
  - Local unit testing support
  - You can run many GridGain nodes on one computer or even in one JVM - debug locally
  - Default peer-to-peer loading with automatic hot-redeployment eliminates costly build process - just recompile locally and run
  - Start as a standalone application or within any hosting environment like JBoss
  - Launch from any IDE (Eclipse, IDEA, NetBeans) in seconds

Scale With GridGain

- Using GridGain often scales linearly so that in most cases you would need just 6 processing nodes to get 6 times performance increase.
- The overall cost of this solution could be only the cost of grid-enabling this task:
  - GridGain is free open-source product
  - Additional computing resources can be easily "drawn" from already existing pool.

GridGain And Multi-Core CPUs

- The raw performance increase per grid node is only part of the advantage multi-core CPUs bring...
- A single processing resource can now support much large amount of processes and threads per process accordingly without usual performance degradation due to excessive thread context switching.
- GridGain provides a consistent programming model whether you run on single, dual or quad-CPUs servers
  - Migration from one type to another 100% transparent.

Factorial Example

```
public class Factorial {
    public static BigInteger factorial(int n){
        BigInteger n = BigInteger.ONE;
        for (int i = BigInteger.ONE; i <= n ; i++)
            n = n.multiply(BigInteger.valueOf(i));
        return n;
    }

}
```

- This method has O(factorial(n)) – higher than exponential order.
- Can we use computational grid here? how ?
  - » ***Split*** n to to several ranges and let each node multiple its range internal numbers
  - » ***Aggreagate*** results by multiply all results from nodes

Grid Instance And Topology

- ***Grid instance*** – is a single runtime named instance of GridGain. ***GridFactory*** class is responsible for starting grid instances.

- ***Grid Topology*** –Grid topology defines what ***grid nodes*** are available for a given ***grid job***. Topology SPI is a centralized place that is responsible for determining what nodes are available to a given ***grid task***.

Grid Task & Grid Job

- ***Grid Task*** – is a main deployment unit in GridGain. Users deploy and execute grid tasks either explicitly (using api) or implicitly (using annotations).

- ***Grid job*** – defines an executable in a Grid Node. Grid job travels to remote nodes for execution.

Task Life Cycle

- GridTask is responsible for:
  - Splitting business logic into multiple grid jobs, **returns a jobs mapped to nodes**.
  - Receiving results from individual grid jobs executing on remote nodes (**result** API). The task can wait for more results, reduce results received so far, or failover a job to another node.
  - Reducing (**aggregating**) received jobs' results into final grid task result. (reduce API).

Task-Session

- *Grid Task Session* – During the entire grid task execution GridGain provides grid task session that is available for grid task and grid jobs instances.
- Has 2 main features:
  - Attribute and checkpoint management.
- Imagine you need to compress a very large file. You need to synch all nodes with repetitions while processing.

Convenient Adapters

- In most homogeneous environments where all nodes are equally suitable for executing grid job use ***GridTaskSplitAdapter***:
  - » Jobs can be randomly assigned to available grid nodes
  - » *GridTaskSplitAdapter.split(gridSize, arg)* – takes given argument and splits it into a collection of GridJob using provided grid size
  - » Implements automatic fail-over to another node if remote job has failed due to a node crash or due to job execution rejection.

Hello JBUG Demo

Loaders

- GridGain was designed to fully "blend" into "*hosting environment*" using SPI implementations.
- Inegration is done using "Loaders"
  - » Provide basic boilerplate code for starting GridGain in various environments.
  - » Usually a loader is given a path of Spring configuration file which should be used for startup.

Hosting Enviroments

- Jboss / WebLogic / Websphere /GlassFish- GridGain can be loaded within these AS and integrate into their native logging, JMX, discovery and communiation.

- JUnit - Takes your regular JUnit TestSuite and runs it in parallel on remote nodes.
- AspectJ - GridGain comes with native integration with AspectJ (libraries are shipped with GridGain)

Hosting Enviroments Cont.

- Spring - GridGain ships with Spring 2.0 as it's used for default configuration implementation. It also support for Spring AOP for grid enabling.

- Coherence/GigaSpaces- GridGain comes with these data-grids implementation for discovery and checkpoints. Checkpoints (intermediate states saved by jobs during their executions) fits perfectly for these data grids solutions..

GridGain JBoss Loader

- JBoss loader is used to start GridGain within JBoss as a JBoss service. In web app you must either:
  - Use “Jboss unified classs loader” OR
  - Copy all GridGain dependencies jars to WEB-INF/lib
- jboss-service.xml has a configuration parameter pointing to Spring XML configuration.
- At startup, GridGain JBoss loader will look for the Spring configuration XML file specified in jboss-service.xml.

GridGain as Jboss service

```
<server>
  <classpath codebase="file:/${GRIDGAIN_HOME}/gridgain-1.6.1.jar"/>
  <classpath codebase="file:/${GRIDGAIN_HOME}/libs" archives="*"/>
  <mbean code="org.gridgain.grid.loaders.jboss.GridJbossLoader"
                                          name="gridgain:service=loader">
    <attribute name="ConfigurationFile">
      config/default-spring.xml
    </attribute>
  </mbean>
</server>
```

```
<server>
  <!--This loader uses a unified class loader as the class loader rather
than the tomcat specific class loader.-->
  <attribute name="UseJBossWebLoader">true</attribute>
</server>
```

Jboss – GridGain Demo

Summary

- If you’re able to express your problem in terms of ***“Map / Reduce”*** operations, you can scale out to many nodes and solve large problems using GridGain.

- GridGain is a “computational-grid” package but provide integration to “data-grids” – ***GigaSpaces*** and ***Oracle Coherence*** as well as application Servers (i.e. JBoss, WebSphere, WebLogic), Spring and JUnit.

Q & A

Thank You

yanai@tikalk.com