GridGain: One Compute Grid, Many Data Grids

Thursday, September 25, 2008 at 2:02AM

Todd Hoff in Grid, Product, cloud

GridGain was kind enough to present at the September 17th instance of the Silicon Valley Cloud Computing Group. I've been curious about GridGain so I was glad to see them there. In short GridGain is: *an open source computational grid framework that enables Java developers to improve general performance of processing intensive applications by splitting and parallelizing the workload*. GridGain can also be thought of as a set of middleware primitives for building applications. GridGain's peer group of competitors includes GigaSpaces, Terracotta, Coherence, and Hadoop.

The speaker for GridGain was the President and Founder, Nikita Ivanov. He has a very pleasant down-to-earth way about him that contrasts nicely with a field given to religious discussions of complex taxomic definitions. Nikita first talked about cloud computing in general. He feels Java is the perfect gateway for cloud computing. Which is good because **GridGain only works with Java**. The Java centricity of GridGain may be an immediate deal killer or a non-issue for a Java shop. Being so close to the language does offer a lot of power, but it sure sucks in a multi-language environment.

Nikita gave a few definitions which are key to understanding where GridGain stands in the grid matrix:

- **Compute Grids**: parallel execution.
- **Data Grids**: parallel data storage.
- **Grid Computing**: Compute Grids + Data Grids
- **Cloud Computing**: datacenter + API. The key is automation via programmability as a way to deploy applications. The advantage is a unified programming model. Build an application on one node and you can run on many nodes without code change. Moving peak loads to the cloud can give you a 10x-100x cost reduction. Cloud computing poses a number of challenges: deployment, data sharing, load balancing, failover, discovery (nodes, availability), provisioning (add, remove), management, monitoring, development process, debugging, inter and external clouds (syncing data, syncing code, failover jobs). Nakita talked some about these issues, but he didn't go in-depth. But he showed a good understanding of the issues involved so I would be inclined to think GridGain handles them well.

The cloud computing section is new to the standard GridGain presentation. GridGain is moving their grid into the cloud with new features like a cloud management layer available in Q1 2009. This move competes with GigaSpaces early move to the cloud with their RightScale partnership. It's a good move. Like peanut butter and chocolate, grids and clouds go better together. Grids have been under utilized largely because of infrastructure issues. A cloud platform makes it is to affordably grow and manage grids, so we might see an uptick in grid adoption as clouds and grids hookup.

GridGain positions themselves as a developer centric framework according to their analysis of cloud computing in Java:

- **Heavy UI oriented**. These types of applications or framework usually provide UI-based consoles, management applications, plugins, etc that provide the only way to manage resources on the cloud such as starting and stopping the image, etc. The key characteristic of this approach is that it requires a substantial user input and human interaction and thus they tend to be less dynamic and less on-demand. Good examples would be RightScale, GigaSpaces, ElasticGrid.
- **Heavy framework oriented**. This approach strongly emphasizes dynamism of resource management on the cloud. The key characteristic of this approach is that it requires no human interaction and all resource management can be done programmatically by the grid/cloud middleware - and thus it is more dynamic, automated and true on-demand. Google App Engine (for Python), GridGain would be good examples.

I think there's a misunderstanding of RightScale here. The UI is to configure the automated system, not manage the system. The automated system monitors and responds to events without human interaction. Won't their automated cloud layer have to do something similar? To bootstrap any complex system out of the mud of complexity a helpful UI is needed. The framework approach of GridGain's infrastructure is developer friendly, but that won't fly for external management within the cloud.

GridGain's True Nature: One Compute Grid, Many Data Grids

With these definitions in place we can now learn the secret of Grid Gain: **One Compute Grid, Many Data Grids**.

Ding! Ding! Ding! Once I understood this I understood Grid Gain's niche. GridGain has focussed on making it dead simple to distribute work across a compute grid. It's a job management mechanism. GridGain doesn't include a data grid. It will work against any data grid. For some reason this fact was something I'd never pulled out of the noise before. And when I would read Nakita's blog with all the nifty little code samples I never really appreciated what was happening. Yes, I'm just that dumb, but I also think Grid Gain should expose the magic of what's going on behind the scenes more rather than push the simple 30-second-lets-write-code-live style demo. Seeing the mechanics would make it easier to build a mental model of the value being added by GridGain.

Transparent and Low Configuration Implementation of Key Features

A compute grid is just a bunch of CPUs calculations/jobs/work can be run on. As a developer problem are broken up into smaller tasks and spread across all your nodes so the result is calculated faster because it is happening in parallel. GridGain enthusiastically supports the MapReduce model of computation.

When deploying a grid a few key problems come up:

- How do you get your code to all nodes? Not just the first time, but every time a JAR file changes how distributed across all nodes?
- How do all the other nodes find each other when they come up? Clearly for work to be sent to nodes someone must know about them.
- How are jobs distributed to the nodes? Somehow jobs must be sent to a node, the calculations made, and the results assembled.
- How are failures handled? Somehow when a node goes down and new nodes come on-line work must be rescheduled.
- How does each node get the data it needs to do its work? Scalable computation without scalable data doesn't work for most problems.

Much of the drama is lost with GridGain because most of these capabilities almost are implemented almost transparently.

Discovery happens automatically. When nodes come up they communicate with each other and transparently form a grid. You don't see this, it just happens. In fact, this was one of GridGain's issues when porting to the cloud. They used multicast for discovery and Amazon doesn't support multicast. So they had to use another messaging service, which GridGain supports doing out-of-the box, and are now working on their unicast own version of the discovery service.

Deploying new code is always a frustrating problem. Over the same transparently formed grid, code updates are transparently auto deployed on the grid. Again, this is one of those things you see happen from Eclipse and it loses most of the impact. It just looks like how it's supposed to work, but rarely does. With GridGain you do a build and your code changes are automatically sent through to each node in the grid. Very nice.

To mark a method a gridified an annotation (or an API call) is used:

```
@Gridify(taskClass = GridifyHelloWorldTask.class, timeout = 3000)
public static int sayIt(String phrase) {
  // Simply print out the argument.
  System.out.println(">>> Printing '" + phrase + "' on this node from grid-enabled method.");
  return phrase.length();
}
```

The task class is responsible for splitting method execution into sub-jobs. For a full example go here.

The @Gridify annotation uses AOP (aspect-oriented programming) to automatically "gridify" the method. I assume this registers the method with the job scheduling system. When the application comes up and triggers execution the method is then scheduled through the job scheduling system and allocated to nodes. Again, you don't see this and they really don't talk enough about how this part works.

Notice how so much complexity is nicely hidden by GridGain with very little configuration on the developer's part. There aren't a billion different XML files where every single part of the system has to be defined ahead of time. The dynamic transparent nature of the core features make it simple to use.

Integrating with the Data Grid

We haven't talked about data at all. If you are just concerned with a program like a Monte Carlo simulation then the compute grid is all you need. But most calculations require data. Where does your massive compute grid pull the data from? That's where the data grid comes in.

A data grid is *the controlled sharing and management of large amounts of distributed data*. GridGain leaves the data grid up to other software by integrating with packages like, JBoss Cache, Oracle Coherence, and GigaSpaces. Remember *One Compute grid, Many Data Grids*. GridGain accesses the data grid through an API so you can plug in any data grid you want to support with a little custom code.

Google and Hadoop use a distributed file system (DFS) as their data grid. This makes sense. When you need to feed lots of CPUs the data can't come from a centralized store. The data must be parallelized and that's what a DFS does. A DFS splats data across a lot of spindles so it can be pulled relatively quickly by lots of CPUs in parallel.

Other products like Coherence and GigaSpaces store data in an in-memory data grid instead of a filesytem. Serving data from memory is faster, but you are limited by the amount of memory you have. If you have a petabyte of data your choice is clear, but if your problem is a bit smaller than maybe an in-memory solution would work. The closer data is to the business logic the better performance will be. GridGain controls job execution while the data grid is responsible for the availability and integrity of the data.

GridGain doesn't care what data grid you use, but your choice has implications for performance. A compute grid and an in-memory data grid in the same cloud will smoke configurations where the data grid comes from disk or is located outside the cloud.

GridGain is Linearly Scalable for a Pure CPU Benchmark

The good folks at GridDynamics are doing some serious cloud testing of different products and different clouds. They did a test Scalability Benchmark of Monte Carlo Simulation on Amazon EC2 with GridGain Software that found GridGain was linearly scalable to 512 nodes in Amazon's EC2. A Monte Carlo simulation is a CPU test only, it does not use a data grid. A data grid based test would be more useful to me as everything changes once large amounts of data start flying around, but it does indicate the core of GridGain is quite scalable.

Wrapping Up

Grid products like Coherence and GigaSpaces include both compute grid and data grid features. Why choose a compute grid only system like GridGain when other products include both capabilities? GridGain might say they win business on the quality of their compute grid, excellent support and documentation, and the ability to cleanly integrate into almost any existing ecosystem through their well thought out API abstraction layer and their out-of-the-box support for almost every important Java framework. Others may counter performance is far better when the business logic and the job management are integrated. All interesting issues to tradeoff in your own decision making process. GridGain is free as their business model is based on providing support and consultation. A non-starter for many is the Java-only restriction. What is unique about GridGain is how easy and transparent they made it to use and deploy. That's some thoughtful engineering.

Related Articles

- Gridify Blog
- Ten Useful Gridgain How-To Tips
- 10 Reasons to Use GridGain
- What is Grid Gain?
- Developers Productivity: Unsung Hero of GridGain
- GridGain vs Hadoop
- Cameron Purdy: Defining a Data Grid
- Compute Grids vs. Data Grids

http://highscalability.com/blog/2008/9/25/gridgain-one-compute-grid-many-data-grids.html