A Java-based Programming Environment for the Grid: **Jojo**

Hidemoto Nakada
National Institute of Advanced Industrial Science and Technology (AIST) / Tokyo Institute of Technology
hide-nakada@aist.go.jp

Satoshi Matsuoka
Tokyo Institute of Technology/ National Institute of Information
matsu@is.titech.ac.jp

Satoshi Sekiguchi
National Institute of Advanced Industrial Science and Technology (AIST)
s.sekiguchi@aist.go.jp

Abstract

This poster introduces a java-based programming environment for the Grid, called **Jojo**. Jojo is a distributed programming environment implemented in Java. It is suitable for a grid environment consists of cluster of clusters. Jojo provides several features, including secure remote invocation using Globus GRAM, intuitive message passing API suitable for parallel execution and automatic user program staging. Jojo helps users to construct their own parallel-distributed application on the Grid.

In the poster, we show design and implementation of Jojo, its programming API and a working program example. We also show preliminary performance evaluation result.

1 Background

Commodity PC clusters are spreading everywhere. Cluster of small clusters will be the typical configuration of the Grid in the near future. While cluster of clusters are cheap and compact, it has a drawback. Due to the IP address exhaustion and security reason, cluster nodes have to be sitting inside a firewall and assigned private addresses. It means that, nodes of different clusters cannot communicate with directly. Existing Grid middleware including MPICH-G2[1] cannot work well in this kind of environment. We need some middleware that can cope with such environment.

Another important issue is the staging of the user code and system code. To ease the burden of installing system codes and user programs on every-single node, automatic staging is needed.

2 Design of Jojo

System Structure Jojo is designed to the hierarchical Grid environment consists of cluster of clusters. Jojo has cascading invocation capability and forms a hierarchal processor tree. It automatically route messages between arbitrary node pair in the whole tree.

Programming Model There are two well-known programming models for parallel communication libraries. One is parallel object model that is adopted by a lot of systems including the Sun's RMI. Another model is SPMD (single program multiple data) model that is mainly adopted by message passing systems such as the MPI. Jojo's programming model is a hybrid of these two models. In Jojo, each node executes just one representative object and it sends and receives messages to/from other nodes. Users cannot create new representative object at runtime.

In Jojo, each node sends and receives single object. While sending message is explicit, message receptions are performed implicitly; namely, there is no `recv` method. Users just define receiving handler for the message. The message handler will be invoked in a separate thread upon the arrival of the message.

Cascading Invocation and code staging To construct a tree of processors, Jojo has a cascading invocation capability. Adding to it, Jojo has a capability to stage user code and system itself. With this capability, users do not need to do any installation sequence. All you need is the Java VM on the cluster nodes.

3 Implementation

3.1 Programming API

A program for Jojo consists of several classes that extend abstract class Code. In Code class, users can uses support classes such as Node and Message.

Code class Abstract class Code is defined as shown in Fig.1. Siblings, descendants and parent point nodes in the same level, upper level and lower level, respectively. The init method initializes the object taking a Map object that contains initialization information. The start method does the real job. The method will never be called before the init method is called. The handle method is the handler that will be invoked when messages arrive to the node. The method is run in a newly created thread. Therefore, if several messages arrive continuously, several threads will execute the handle method simultaneously. It means you can do something that will take a long time in the handler without disturbing other message handlings. On the other hand, you have to perform proper mutual exclusive control to access shared resources.

```
abstract class Code{
  Node [] siblings;    /** sibling nodes */
  Node [] descendants; /** child  nodes */
  Node    parent;      /** parent node */
  int     rank;  /** rank within the brothers */
  /** initialization  */
  public void init(Map arg);
  /** body  */
  public void start();
  /** handling the incoming messages */
  public Object handle(Message mes);
}
```

Figure 1: Code Class

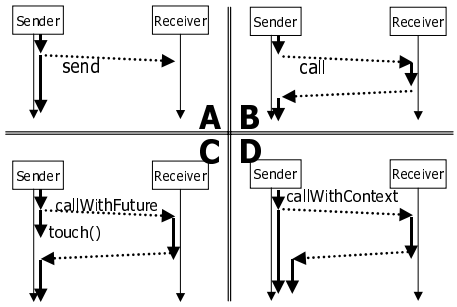

Figure 2: Communication Modes

3.1.1 Code

Node class Code class objects communicate each other invoking methods of Node class. Node class provides following four communication methods to enable flexible asynchronous communication.

- `void send(Message msg)`
  Simply sends a message object, then return immediately(Fig.2:A).
- `Object call(Message msg)`
  Sends a message object and wait for a reply object, then returns the reply (Fig.2:B).
- `Future callFuture(Message msg)`
  Sends a message object and immediately returns with Future object. Future object is a kind of wrapper for the reply object. When the `touch()` method is invoked, it will return the reply object if it is ready. If not, it will block till the reply object arrives (Fig.2:C).
- `void callWithContext(Message msg, Context context)`
  Takes an object that implements Context interface as the second argument. This method sends a message object and returns immediately. When the reply object arrives, it will invoke a thread and calls run method of the object with the reply object (Fig.2:D).

3.2 Remote Invocation

To reduce the burden of installation, Jojo system stages not only user programs, but also Jojo system itself. For this purpose, we used bootstrap loader **rjava** [2]. The Jojo invocations steps are as follows;

1. Rjava bootstrap server which includes special-made class loader is transferred to the server as a jar file.
2. Invoke a bootstrap server and establish connection between the client and the bootstrap server. The client is responsible to read and transfer of class files needed by the bootstrap class loader.
3. The bootstrap server on the remote node invokes Jojo system classes. These classes are automatically loader from the client local file system via the rjava client.
4. User program on the Jojo system also will be loader from the client local file system. For the invocation, users can use Globus[3] GRAM/GASS, rsh/rcp, and ssh/scp.

4 Conclusion and Future work

In the poster, we introduce a java-based programming environment for the Grid, called **Jojo**. Due to the space limitation, we cannot show and discuss on program sample and performance evaluation result in this abstract. They will appear in the poster.

Future work are following:

- We will prove scalability of Jojo through implementation of several combinatorial optimization problems using **jPoP**[4], which is implemented on top of the Jojo.
- In the current implementation, sibling nodes cannot communicate each other directly; instead, they communicate through the parent node. This is because; authentication issue in some environment prohibits direct communication among siblings However, in certain situation it is possible to perform direct communication. We will implement it to improve total performance.

References

[1] Roy, A., Foster, I., Gropp, W., Karonis, N., Sander, V. and Toonen, B.: MPICH-GQ: Quality-of-Service for Message Passing Programs., *Proc. of the IEEE/ACM SC2000 Conference* (2000).

[2] Sohda, Y., Nakada, H., Ogawa, H. and Matsuoka, S.: Implementation of Portable Software DSM in Java, *Proc. of JavaGrande 2001*, pp. 163–162 (2001).

[3] Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications and High Performance Computing*, Vol. 11, No. 2, pp. 115–128 (1997).

[4] jPoP: A parallelization framework for combinatorial optimiation problems. http://ninf.apgrid.org/jpop.