A Java-based Programming Environment for the Grid: Jojo

Hidemoto Nakada (AIST / Tokyo-tech)
Satoshi Matsuoka (Tokyo-tech / NII)
Satoshi Sekiguchi (AIST)
Background

Grid and cluster are popularized
- Clusters as Grid nodes

Problem1: All the cluster nodes cannot have global IP
- Existing middleware does not handle private IP nodes
  - Ex. MPICH-G2, C.f. NXProxy [Tanaka]

Problem2: Client scalability
- Client node cannot handle hundreds of Servers

Hierarchical Grid
Goal

Provide a programming environment that fits to hierarchical Grid

- Works with private addressed clusters
- Scales to hundreds of nodes

Java based Grid middleware Jojo
Jojo (1)

Communication with tree structure

- Private addressed cluster ready
- Protocol: Globus GRAM, ssh, rsh
Jojo (2)

- Adopt Java language
  - Code portability
    - Good for heterogeneous environment
  - Integrated Thread support
    - Good for latency hiding
  - Lot of libraries are available
    - XML, Web related, network communication
Jojo (3)

- Executes each codes on each nodes
  - Each nodes can communicate with its parent, siblings, children.
- Automatically downloads user programs, and Jojo system program.
  - Avoids system version miss-match
  - Requires Java VM only on the cluster nodes
Programming Model

Requirements

- Simple, intuitive
- Flexible message passing
  - Latency hiding with threaded programming

- Complex asynchronous communication can be programmed easily
Programming model of Jojo

- Each node executes its own code
  - Subclass of Code class
  - SPMD

- Object based messaging
  - Incoming messages are handled by handler method
  - RPC style call is supported
    - Several message transfer modes are supported
API (1) Code

abstract class Code{
    Node [] siblings;  /** Brothers */
    Node [] descendants; /** children */
    Node parent;       /** parent */
    int rank;          /** order in the brothers */
    /** initialize */
    public void init(Map arg);
    /** actual task */
    public void start();
    /** handler to handle incoming messages */
    public Object handle(Message mes);
}

public interface Node {
    /** non-blocking send; do not wait for reply */
    void send(Message msg) ;

    /** blocking call; wait for reply */
    Object call(Message msg) ;

    /** non-blocking call; do not wait for reply.
     * returns Future to synchronize the reply. */
    Future callFuture(Message msg) ;

    /** non-blocking call; execute unnable */
    void callWithContext(Message msg, Context context) ;
}
public class Message implements Serializable{
    /** message id */
    public int tag;

    /** message contents */
    public Serializable contents;
}

API (3) Message
Transmission mode (1) send

Sender

Receiver

send
Transmission mode (2) blocking call

Sender

call

Receiver
Transmission mode(3) Future

Sender

Receiver

\[ f = \text{node.callFuture}(o); \]

\[ \text{out} = \text{f.touch}(); \]
Transmission mode(4) with Context

Sender

Receiver

callwithContext

The context is executed in separate thread
Starting up Jojo Program

- Configuration file includes node and code configuration.
- Properties file is to specify the
- All the programs including Jojo itself automatically staged from the client
  - Boot strap server rjava
Bootstrapping by rjava
Configuration File

- Described in XML
- Represent hierarchical structure

```xml
<!ELEMENT node (code?, invocation?, node*)>
<!ATTLIST node host CDATA #REQUIRED>
<!ELEMENT code (#PCDATA)>
<!ELEMENT invocation EMPTY>
<!ATTLIST invocation
  javaPath CDATA #IMPLIED
  rjavaProtocol CDATA #IMPLIED
  rjavaRsh CDATA #IMPLIED
  rjavaRcp CDATA #IMPLIED
  xtermDisplay CDATA #IMPLIED
  xtermPath CDATA #IMPLIED
>
```
Sample configuration file

```xml
<nodes>
  <node host="root">
    <code>PiMaster</code>
  </node>
  <node host="default">
    <code>PiWorker</code>
    <invocation
      javaPath="java"
      rjavaJarPath="/tmp/rjava.jar"
      rjavaProtocol="ssh"
      rjavaRsh="ssh"
      rjavaRcp="scp"/>
  </node>
  <node host="pad00"/>
  <node host="pad01"/>
  <node host="pad02"/>
  <node host="pad03"/>
</nodes>
```
Program input

- Specifies properties file
  - Properties are passed to all the Codes

> java silf.jojo.Jojo CONF_FILE PROP_FILE
Sample Program

Calculate Pi

- Randomly generates large number of points in a square
- Count the number in the arc
- Calculate Pi from the probability
- Master – Worker model
  - Dynamic load balancing

\begin{align*}
\text{times} &= 100000 \\
\text{divide} &= 10
\end{align*}
public class PiMaster2 extends Code{

    ...

    synchronized public Object handle(Message msg) throws JojoException{
        if (msg.tag == PiWorker.MSG_TRIAL_REQUEST){
            long [] pair = (long[])(msg.contents);
            doneTrial += pair[0];
            doneResult += pair[1];
            if (doneTrial >= times){
                synchronized (this) {done = true; notifyAll();}
                return new Long(0);
            } else
                return new Long(perNode);
        } else
            throw new JojoException("cannot handle the message: " + msg);
    }
Sample Program (Worker)

```java
public class PiWorker2 extends Code{

    public void start() throws JojoException{
        long trialTimes = 0, doneTimes = 0;
        while (true){
            Message msg =
                new Message(MSG_TRIAL_REQUEST,
                        new long[]{trialTimes, doneTimes});
            trialTimes =
                ((Long)(parent.call(msg))).longValue();
            if (trialTimes == 0) break;
            doneTimes = trial(trialTimes);
        }
    }

    private long trial(long trialTimes) {
        long counter = 0;
        for (long i = 0; i < trialTimes; i++) {
            double x =
                random.nextDouble();
            double y =
                random.nextDouble();
            if (x * x + y * y < 1.0)
                counter++;
        }
        return counter;
    }
}
```
Preliminary Evaluation

Throughput measurement in LAN and WAN
- AIST and Titech

GSI and SSH
- GSI uses pure-Java SSL
- SSH uses external program
  - Written in C

Throughput: 10 Mbyte/s Latency: 7 ms
Throughput: 54.3 Mbyte/s Latency: 0 ms

Client A
Linux PC Athlon 1.2GHz
Throughput: 10 Mbyte/s Latency: 7 ms

Client B
Linux PC Pentium III 1.4GHz
Throughput: 54.3 Mbyte/s Latency: 0 ms

Server
Linux PC Pentium III 1.4GHz

Gigabit Ether
50 miles

TITECH

AIST
Result (WAN)

- Bandwidth of the link is 10Mbyte/s
- 70-80% of the bandwidth
- SSH is faster slightly
Result (LAN)

- Bandwidth of the link is 54 Mbyte/s
- GSI is much slower than SSH (2/3 of SSH)
Master-Worker evaluation

Compare 2-layered and 3-layered

2layers model

3layers model
Environment

- CATV + Wireless LAN
- Master-worker PI
  - $10^4$ tasks
  - Execution time for a single task is around 8ms

Client

Throughput:
- to Cluster: 0.07 Mbyte/s
- from Cluster: 0.17 Mbyte/s
- Latency: 150ms

Cluster

Throughput:
- Gigabit Ether: 54.3 Mbyte/s
- Latency: 0ms

Management Node

Linux in VMWare
- Pentium 4 1.8GHz

Linux PC
- Pentium III 1.4GHz
- Dual x (32 + 1)
Master-Worker result

- Scales up to 8 nodes
- Scales up to 16 nodes
- Much faster than 2-layered
Discussion

- Data size for each task is just few bytes
  - Time for data transfer time is negligible
  - Latency

- Execution time for each task is just 8ms
  - Not suitable for master-worker execution
    - As shown in the 2-layered model score
  - Still can be effectively executed in 3-layered model
Summary

- Proposed a grid middleware Jojo
- Shows sample programs
- Shows preliminary evaluation results
  - Pingpong
  - Master-worker