Field based process migration

Introduction

A process is an instance of a running program that is being executed. It contains the program code and data. Process migration is transferring the process from one system to another system either manually or through middleware. When process is migrated to another system, it should also carry the program code and data to the target system. Process migration is a key concept in Grid computing. WIKI defines Grid computing as a form of distributed computing where grid represents “super virtual computer” composed of many loosely coupled computers acting together to perform very large tasks.

Process migration benefits

  a) Fault resilience - If current computing nodes experiences a failure
  b) Load balancing – If load on current computing node increases

AgentTeamwork is one of the middleware developed by Professor Munherio Fukuda to perform process migration for Java based applications. Current implementation of framework supports process migration based on static list of computing nodes. The load on a computing node can vary from time to time, so process migration based on static list of nodes won’t give the best result in terms of performance.

Autonomic computing is the emerging technology where industry is making significant investments. Autonomic computing refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes whilst hiding intrinsic complexity to end users. IBM defines the Autonomic computing characteristics as follows:

  ➢ Self-Configuration: Automatic configuration of components;
  ➢ Self-Healing: Automatic discovery, and correction of faults;
  ➢ Self-Optimization: Automatic monitoring and control of resources to ensure the optimal functioning with respect to the defined requirements;
  ➢ Self-Protection: Proactive identification and protection from arbitrary attacks.

This motivates us to enhance the AgentTeamwork further to make its process migration algorithm based on computing nodes load at runtime. Also motivates us to make use of additional computing nodes available in other network segments.

AgentTeamwork dispatches mobile agents to collection of computing nodes to perform a user submitted job, which maintains high availability of the computing resources. A mobile agent is a process that can transport its state from one environment to another, with its data intact, and be capable of performing appropriately in the new environment. AgentTeamwork monitors the execution of mobile agent at a different machine and takes its snapshot periodically. This helps to move and resume the mobile agent on a different machine from its latest snapshot upon an accidental crash. This framework offers mobile agent execution platform, error-recoverable socket and mpiJava API libraries.
The main goals of this research are

1) Migrate the Process based on dynamically evaluated criteria - we can run a program in each participating node to broadcast its local system information including CPU and Memory information as UDP messages. Then an agent responsible for spawning a process can sort these nodes based on user specified criteria and makes the destination node(s) list to migrate the process.

2) Make use of computing nodes available in other networks.

We have implemented a couple of mobile agents such as Potential Field (PF), Commander and Sentinel agents based on this framework to accomplish field based migration. This paper presents the implementation techniques of these agents, algorithms to determine the best computing nodes and performance comparison between best and worst computing nodes, determined by this algorithm using Java Wave2DMPI application.

**System overview**

**PFAgent**

PFAgent is one of the mobile agents which will be running in all participating computing nodes as a daemon process. This will broadcast resource information such as number of CPUs, number of cores, number of users, CPU load and memory information as UDP messages. It also returns the best computing nodes based on nodes computing capability (referred as rank). Since it broadcasts as UDP message there is no need to maintain the static list of computing nodes.

Factors affecting computing capability are:

- Number of CPUs
- Number of CPU Cores
- CPU Speed
- CPU Load - is a measure of the amount of work that a computer system performs.
- Free memory (Mf)
- Total available memory
- Network bandwidth - bandwidth in bit/s represents the maximum throughput of a computer network.

From the above factors we calculate ranks for entities like CPU, memory_free, memory_load and n/w bandwidth. Finally, a rank will be assigned to each node within the subnet. The higher ranking node will have higher processing power.

\[
\text{Cpu capacity (C)} = \left(\frac{\text{#CPUs} \times \text{#Cores} \times \text{CPU speed}}{1 - \text{CPU load}}\right)
\]

\[
\text{Cpu rank of } N_1 (N_1 C) = \left\{ \frac{N_1 C}{N_1 C + N_2 C + \ldots + N_n C} \right\} \%
\]

\(N_1 C = \text{Node's (N1) CPU capacity}\)
Node's CPU rank.

\[
N_{1c} = \text{memory\_freerank of } N_1 \left( \frac{N_1 M_f}{N_1 M_f + N_2 M_f + \ldots + N_n M_f} \right) \%
\]

\[
N_{1m} = \text{memory\_rank of } N_1 \left( \frac{N_1 M - N_1 M_f}{N_1 M_f + N_2 M_f + \ldots + N_n M_f} \right) \%
\]

\[
N_{1m} = \text{bandwidth\_rank of } N_1 \left( \frac{N_1 B}{N_1 B + N_2 B + \ldots + N_n B} \right) \%
\]

Memory\_rank represents the memory pressure on the node. The higher value represents less memory pressure.

When PFAgent starts, it runs ttcp server thread in all computing nodes. When bandwidth is calculated, each node will treat the next node as server node (sorted based on node name) within the network segment. For example, in a network segment, if we have N1, N2, … Nn nodes while calculating network bandwidth for node N1, N2 will be treated as a server node and will receive data from N1. Based on the ttcp output, network bandwidth will be calculated. For Nn node N0 will be the server node.

CPU, memory and bandwidth ranks play key role in determining resource capability. However CPU rank has more precedence than memory and memory has more precedence than bandwidth rank in determining processing power. For our initial assessment we’ve randomly assigned a weight to these parameters and calculated overall rank of the node as follows.

\[
\text{Overall rank for } N_1 \left( N_1 R \right) = \left\{ \left( N_{1c} \times 0.5 \right) + \left( N_1 M_f \times 0.2 \right) + \left( N_1 M \times 0.1 \right) + \left( N_1 B \times 0.2 \right) \right\}
\]

This weight either needs to be increased or decreased based on the research.
**Investigation Items in Job Migration**

The first item is what resource information PFAgents should use and weight when constructing a computing-resource potential among them. We will use the following formulae 1 - 5 to calculate the performance measure of each PFAgent that receives resource information with its N neighbors. The weight1 through to weight4 parameters will be decided through our research.

The second item is under what conditions a mobile agent should move a job from the current to a next computing node. For instance, a mobile agent could use the following formula 6 where hop overhead needs to be empirically obtained and job_size may be fixed to temperature analysis, (e.g., 20 minutes to 76 minutes based on our single CPU execution experiment.) Needless to say, job_size decreases as the job continues to run from one to another node.

\[
\text{cpu\_rank}_i = \frac{\text{#cores}_i \times \text{#cpus}_i \times \text{speed}_i \times (1 - \text{cpu load}_i)}{\sum_{j=1}^{N} \text{#cores}_j \times \text{#cpus}_j \times \text{speed}_j \times (1 - \text{cpu load}_j)}
\]

\[
\text{Memory\_free\_rank}_i = \frac{\text{memory\_free}_i}{\sum_{j=1}^{N} \text{memory\_free}_j}
\]

\[
\text{memory\_load\_rank}_i = \frac{\text{memory\_free}_i / \text{memory\_total}_i}{\sum_{j=1}^{N} \text{memory\_free}_j / \text{memory\_total}_j}
\]

\[
\text{network\_bandwidth\_rank}_i = \frac{\text{network\_bandwidth}_i}{\sum_{j=1}^{N} \text{network\_bandwidth}_j}
\]

\[
\text{comprehensive\_rank}_i = \text{cpu\_rank}_i \times \text{weight}_1 + \text{memory\_free\_rank}_i \times \text{weight}_2 + \text{memory\_load\_rank}_i \times \text{weight}_3 + \text{network\_bandwidth\_rank}_i \times \text{weight}_4
\]

\[
\text{job\_size} \geq \text{hop\_overhead} + \text{job\_size} \times \frac{\text{rank\_current}}{\text{rank\_next}}
\]

**Commander Agent**

Commander Agent is a type of mobile agent which is responsible for accepting user input and program class file and spawns Sentinel Agent to execute the user program. It waits for the user program to complete its execution and update the results back to the user.

**Sentinel Agent**

Sentinel agent is one of the mobile agents which actually execute the user program. It migrates between computing nodes based on the best computing node returned by PF Agent. The current implementation of migration algorithm checks the best computing node before it starts executing a user program. If it finds a better node than the current node, it will hop to the best node. This way Sentinel Agent will hop to maximum of 3 computing nodes (This is hypothetical number. This number is subject to change based on research outcome). Once it reaches third node, it will start executing the user program.
Research methods

Wave2DMpi is a MPI based Java application which was used to measure the performance evaluation between worst vs. best computing nodes returned by PFAgent. Our initial assessment based on developed prototype returned better results for when this program was executed by best computing nodes.

Syntax: Wave2DMpi [Procnumber] [size] [time] [interval]

We executed the program “Wave2DMpi 3 1000 10000 100” in UW1-320 Linux lab (nodes 1 to 10) and evaluated the performance between best and worst computing nodes. Best computing nodes completed the computation faster than the worst computing nodes.

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Best Computing Nodes</th>
<th>Processing time (Secs)</th>
<th>Worst Computing nodes</th>
<th>Processing Time (Secs)</th>
<th>Gain / Loss (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01, 05 &amp; 03</td>
<td>158.345</td>
<td>07, 09 &amp; 03</td>
<td>167.682</td>
<td>9.337</td>
</tr>
<tr>
<td>2</td>
<td>01, 09 &amp; 06</td>
<td>157.766</td>
<td>07, 00 &amp; 06</td>
<td>166.039</td>
<td>8.273</td>
</tr>
<tr>
<td>3</td>
<td>09, 01 &amp; 05</td>
<td>158.921</td>
<td>07, 00 &amp; 04</td>
<td>163.266</td>
<td>4.345</td>
</tr>
</tbody>
</table>

Conclusion

This preliminary prototype and results indicates that dynamic Field based process migration is better than dispatching process over static list of nodes. This algorithm needs to be further enhanced to adjust the weights of CPU, memory and bandwidth ranks, and to include process migration cost while making the process migration decision. Complex scientific computations such as tsunami predictions, temperature simulation programs will greatly benefit from this research.

Appendix

Source Code : /net/metis/home3/dslab/SensorGrid/FieldBaseMigration/agents

Steps to run the program

1) Compile the source code
   
   /net/metis/home3/dslab/SensorGrid/FieldBaseMigration/agents> compile.sh

2) Navigate to drop folder
   
   cd drop

3) Launch UWPlace and PFAgents
   
   /net/metis/home3/dslab/SensorGrid/FieldBaseMigration/agents/drop> launchagents.sh

4) Wait for few minutes to allow PFAgent to broadcast resource information. Then run user program
/net/metis/home3/dslab/SensorGrid/FieldBaseMigration/agents/drop> runcommand.sh 3 1000 10000 10 best

**Output**
Wave2D.out – will be available at
/net/metis/home3/dslab/SensorGrid/FieldBaseMigration/agents/drop/<<GUID>>/

UWAgent.out - will be available at /net/metis/home3/dslab/SensorGrid/FieldBaseMigration/agents which will have processing time.

**Artifacts**

1) **AgentLauncher.java**

This class launches UWPlace, PFagent on nodes returned by “cat /etc/hosts” command. This uses jsch to launch these agents on local node as well as on remote nodes. Jsch runs the program from user’s home directory.

Note: - Currently it is configured to run only on 10 nodes (whose name is starting with UW1).

**Commands**

```
java -cp *: AgentLauncher uwplace /classpath $(pwd) /command launch /portnumber 12345 /user dslab /password [pwd]

java -cp *: AgentLauncher pfagent /classpath $(pwd) /command launch /portnumber 12345 /user dslab /password [pwd]
```

We can also use this for quitting UWPlace agent.

Note: It uses “killjava.sh” [Which looks up the process name (java) to kill it]. Since Agentlauncher is also a java program we can’t use this to kill agents on current node.

**Command**

```
java -cp *: AgentLauncher uwplace /classpath $(pwd) /currentnode $(hostname) /command quit /portnumber 12345 /user dslab /password [pwd]
```

Note: This won’t exit the agents running on current node. You need to call “killall java” to exit explicitly. Otherwise use the scripts.

2) **PFagent.java**
This class broadcasts local system information as UDP messages also receives UDP messages broadcasted by other nodes. This broadcasts system information every 1 minute. It also calculates the best computing node.

To see the messages broadcasted by PFagent (within the network) run the test program (uses hardcoded port# 12345)

Command: java Test show

To get the best computing node run the following command

Command: java Test getbestnodes

3) BroadcastSocket.java

This is used by PFagent to broadcast system information.

4) CommanderAgent.java

This is run by the user to launch Commander Agent which will in turn launch Sentinel Agent which executes user program.

java -cp UWAgent.jar:. UWAgent.UWInject -p 12345 -j commander.jar localhost \CommanderAgent U_S(pwd)/Wave2DMpi.class_1000_10000_100 USER_dslab PWD_<<password>> SN_3 TEST_BEST

SN_3 = represents the number of computing nodes.
Test_best = To run user program on best computing nodes
Test_worst = To run user program on worst computing nodes.

Before run this command, run “java Test show” to make sure PFagent broadcast messages. Also run command “java Test getbestnodes”. From this go to the best computing node (first one) then launch the commander agent from that one (Test_best) for “Test_worst” launch it from the last node.

5) SentinelAgent.java

SentinelAgent is launched by commander agent. This is responsible for launching user program.

6) Utilities.java

This is a wrapper for Jsch functions.

7) Wave2DMpi.java

This is the user program to be launched by SentinelAgent.

Command

mpdboot –n [procnumber]

prunjava [procnumber] Wave2DMpi [size] [time] [interval]
Note: - before run this command, we need to have mpd.hosts file. The list of entries in mpd.hosts should match the procnumber. Once this is done, we have to run command “mpdallexit”.

8) **Test.java**  
This is a test program for PFAgent.

Commands  
java Test show - To display the UDP messages.  
java Test getbestnodes - To get the best computing nodes.

**Script files**

1) **Killjava.sh**  
This is used to kill a process named “java”. This is utilized by AgentLauncher.java (to kill the UWPlace agents).

2) **Compile.sh**  
a) Deletes & recreates drop folder  
b) Compiles all java files  
c) Makes jar files (Utilities.jar, pfagent.jar, and commnader.jar).  
d) Copies all class files, jar files and script files to drop folder.

*The following script needs to be run from drop folder*

3) **launchagents.sh**  
This launches AgentLauncher to run UWPlace agent and PFAgent.

4) **runcommand.sh**  
This launches CommanderAgent and passes “Wave2DMpi.classes” as a user program argument.