Mahjong Manual

Yuriy Brun
ybrun@usc.edu
http://alum.mit.edu/www/brun

February 26, 2009

1 What is Mahjong?

Mahjong is a distributed software system that uses idle cycles on remote but networked computers to solve NP-complete problems, such as 3-SAT, determining the structure of proteins, optimally allocating resources, and many others. Mahjong is designed using the tile architectural style [1], a biologically-inspired software architectural style that fascilitates distributing computationally intensive and easily parallelizable problems onto large networks in a scalable, privacy-preserving, and fault-tolerant manner. Mahjong is written in Java, is platform independent, and uses PrismMW, a lightweight architecture-aware middleware [2], to ensure its compliance with the tile style.

2 Downloading Mahjong

In order to run Mahjong, every computer that will participate in the computation must run Java. The JDK (Java Development Kit) is available from Sun Microsystems. All other necessary software, such as the PrismMW libraries, comes with the Mahjong package.

The latest version of the Mahjong package can be downloaded from the Mahjong website.

3 Compiling Mahjong

The package you download should already contain the compiled .class files; however, if recompiling the code is simple. Just make sure the PrismMW2_1.jar file is in your classpath (see for help on how to manipulate the classpath) and compile all the include .java files. Mahjong has been tested to compile with both `javac` and `jikes`.

For example, the following command should compile all the source code on Windows (the classpath separator ; is what makes this command Widnows-specific):

```
javac -cp .;PrismMW2_1.jar
   tileStyle\*.java tileStyle\Prism\*.java tileStyle\Prism\events\*.java
```

4 Running a Simple Mahjong Application

Mahjong lets you distribute NP-complete problems onto nodes. In order to run Mahjong, in addition to the binaries, you need a file that describes the problem (.tiles file) and a file that describes the input to the problem (.seed file). The downloadable Mahjong package includes a sample .tiles file (SubsetSum.tiles) for the NP-complete problem *SubsetSum*, and a sample .seed file (18.seed).

4.1 Deploying Mahjong on a Single Machine

Armed with Java, the binaries, and the .tiles and .seed file, you can run Mahjong on a single machine. You first must make sure that the tileStyle directory and PrismMW2_1.jar file are in your classpath. How to set this classpath depends on your platform, but Sun's documentation on classpaths can be quite helpful if you are having problems. To execute the example on a single machine, execute the following command:

```
java tileStyle.TileStyleStarter server localhost 1 SubsetSum.tiles 18.seed 18.out 0
```

This command will execute the sample Mahjong problem on a single computer. The code will run for a little bit, and eventually print something that contains the words "Found solution at". The output of the computation will be stored in the file `18.out`. This output will contain a lot of data about tiles that attach and timestamps on virtually everything that happens. Because many processes are happening in parallel, the "Found solution at" is unlikely to be the last line of the output and you might have to scroll up some to find it. Also, once a node finds a solution, it signs off the network, causing all the other nodes to throw exceptions and complain. Thus the system is likely to "quit ungracefully" by covering the screen with exceptions and possibly hanging Java.

You just deployed Mahjong on a single machine and solved a very simple SubsetSum problem!

While for now I have specified the command-line argument to Mahjong, Section 4.3 will describe those arguments in detail and explain the values they may take on.

4.2 Deploying Mahjong on Multiple Machines

When deploying Mahjong on multiple machines, one machine acts as the server and the rest as clients. All the computers that will work together must be able to reach each other via their TCP/IP interfaces (e.g., ping each other). In other words, they need to all be able to access the same local area network or the Internet. There are a number of connectivity issues that may arise with firewalls, operating systems, or routers that this manual does not discuss.

At the start of the computation, the server waits for all the clients to sign in, and following that, the computation begins. To start the server, execute the following command:

```
java tileStyle.TileStyleStarter server MYIP 2 SubsetSum.tiles 18.seed 18.out 0
```

Note that there is an argument `MYIP` and an argument 2 in the command. The `MYIP` keyword must be replaced by the IP address or hostname of the server. The 2 tells the server that there will be 2 machines (including the server) on this network. The server will sit patiently and wait for the 2nd machine to join before proceeding. Increasing the 2 increases the number of machines the server will use.

On the client machine(s) execute the following command:

```
java tileStyle.TileStyleStarter client MYIP SERVERIP
```

Note that there are arguments `MYIP` and `SERVERIP` in the command. `MYIP` must be replaced with the IP address or hostname of the particular client executing this command. This step is necessary because a computer might have multiple network adapters, multiple TPC/IP interfaces, and multiple IP addresses, and Mahjong needs to know which one to use to access the other computers. Similarly, `SERVERIP` must be replaced with the IP address of hostname of the server, so the client knows whom to contact. If a client starts before the server, it will patiently wait for the server to become available.

Once you run these commands, the client will sign into the server, they'll exchange important information, and the computation will start. The code will run for a little bit, and eventually print something that contains the words "Found solution at" to the output file `18.out` on the server machine. This output will contain a lot of data about tiles that attach and timestamps on virtually everything that happens. Because many processes are happening in parallel, the "Found solution at" is unlikely to be the last line of the output and you might have to scroll up some to find it. The clients will not have output files. Also, once a node finds a solution, it signs off the network, causing all the other nodes to throw exceptions and complain. Thus the system is likely to "quit ungracefully" by covering the screen with exceptions and possibly hanging Java.

You just deployed Mahjong on two or more machines and solved a very simple SubsetSum problem!

4.3 Command-Line Arguments

When Mahjong starts on a computer, it either assumes the role of a *server* or a *client*. The first command-line argument to Mahjong determines that role. When executing as a client, Mahjong expects the following command line:

```
java tileStyle.TileStyleStarter client MYIP SERVERIP
```

Let's examine the arguments:

`MYIP`	The IP address or hostname of the particular client executing this command.
`SERVERIP`	The IP address of hostname of the server.

When executing as a server, Mahjong expects the following command line:

```
java tileStyle.TileStyleStarter server MYIP NUM TILES SEED OUTFILE JOBID
```

Let's examine the arguments:

`MYIP`	The IP address or hostname of the server.
`NUM`	The number of computers, including the server, that will participate in the computation.
`TILES`	The path to a .tiles file that describes the problem being solved.
`SEED`	The path to a .seed file that describes the input to the problem being solved.
`OUTFILE`	The path to the file that Mahjong will use to store output.
`OUTFILE`	An ID for this job (necessary for some cluster distribution protocols).

5 Debugging Information

The are a number of procedures Mahjong follows that may be helpful to understand when debugging the application. This section contains information on some of these procedures.

5.1 Ready, Set, Go Sign In Procedure

Before Mahjong can distribute computation, the server must collect the proper number of clients and distribute some information to those clients. During the set up process, the clients and the server go through the "ready, set, go" process:

Ready: Each client sends a "sign in" event to the server. Once the client sends this event, it is ready. Once the server receives a sufficient number of such events, it is ready.

Set: The server creates data (addresses map) needed to send to each client and sends it out. Every client and server create the appropriate number of virtual nodes and an incoming connection port on each virtual node. Each client sends the server an "I am set" event. Once the server receives a sufficient number of such events and creates its own virtual nodes and incoming connections, it is also set.

Go: The server tells everyone that everyone is set and each virtual node tries to create connections to all the other virtual nodes. Once each client finishes connections for all its virtual nodes, it sends an "I am go" event to the server. Once the server receives a sufficient number of such events and once its virtual nodes create all their connections, it is also go and starts the computation.

References

[1] Yuriy Brun and Nenad Medvidovic. Preserving privacy in distributed computation via self-assembly. Technical Report USC-CSSE-2008-819, Center for Software Engineering, University of Southern California, 2008.

[2] Sam Malek, Marija Mikic-Rakic, and Nenad Medvidovic. A style-aware architectural middleware for resource-constrained, distributed systems. *IEEE Transactions on Software Engineering*, 31(3):256–272, 2005.