Ibis MPJ User's Guide

http://www.cs.vu.nl/ibis

November 20, 2009

1 Introduction

The MPJ programming interface has been defined by the Java Grande forum to provide MPI-like message passing for Java applications. Ibis MPJ is a pure-Java implementation of this interface, and delivers high-performance communication, while being deployable on various platforms, from Myrinet-based clusters to grids.

1.1 Virtual Topologies

Depending on the underlying algorithm, logical communication structures like hypercubes or graphs, may appear, called virtual topologies in MPJ. At the moment, virtual topologies are not supported by Ibis MPJ. Only linear rank numbering is supported.

1.2 Intercommunication

It is not possible, that two processes, both members of different groups, can exchange messages. Ibis MPJ does not support MPI-like inter- communication at the moment. To achieve intercommunication, a new group, e.g. derived from the group of `MPJ.WORLD_COMM`, has to be constructed containing the processes that are supposed to communicate to each other.

1.3 Derived Datatypes and Multidimensional Arrays

Since Java provides derived datatypes natively, by using Java objects, there is no real need to implement derived datatypes in Ibis MPJ. Nevertheless contigous derived datatypes are supported to obtain the functionality of the reduce operations `MPJ.MINLOC` and `MPJ.MAXLOC`, which need at least a pair of values inside a given array. At the moment Ibis MPJ supports one-dimensional arrays. Multidimensional arrays may be sent as an object, but in place receive is not possible in this case.

2 Compiling an example

The Sor example application for MPJ is provided with the Ibis MPJ distribution, in the `examples` directory, which also contains various other examples. For convenience, the examples are already compiled.

If you change the example, you will need to recompile it. This requires the build system `ant`[1]. Running `ant` in the examples directory compiles the examples, and rewrites the class files for use with Ibis MPJ.

If, for some reason, it is not convenient to use *ant* to compile your application, or you have only class files or jar files available for parts of your application, it is also possible to compile your application to class files or jar files, and then process those using the *mpjc* script. This script can be found in the Ibis MPJ bin directory. It takes either directories, class files, or jar files as parameter, and processes those, possibly rewriting them. In case of a directory, all class files and jar files in that directory or its subdirectories are processed. The command sequence

```
$ cd $MPJ_HOME/examples
$ mkdir tmp
$ javac -d tmp -g \
     -classpath ../lib/mpj-2.2.jar \
     src/*/*.java
$ ../bin/mpjc -cp tmp tmp
$ mkdir lib
$ ( cd tmp ; jar c . ) > lib/mpj-examples.jar
$ rm -rf tmp
```

creates a `lib` directory and stores the resulting class files there, in a jar-file called `mpj-examples.jar`. The `MPJ_HOME` environment variable must be set to the location of the Ibis MPJ installation.

Running `mpjc` is not obligatory, but it may improve serialization performance, i.e. the writing and reading of Java objects.

3 An Ibis MPJ run

Before discussing the running of an Ibis MPJ application, we will discuss services that are needed by the Ibis communication library.

3.1 The pool

A central concept in Ibis is the *Pool*. A pool consists of one or more Ibis instances, usually running on different machines. Each pool is generally made up of Ibises running a single distributed application. Ibises in a pool can communicate with each other, and, using the registry mechanism present in Ibis, can search for other Ibises in the same pool, get notified of Ibises joining the pool, etc. To coordinate Ibis pools a so-called *Ibis server* is used.

3.2 The Ibis Server

The Ibis server is the Swiss-army-knife server of the Ibis project. Services can be dynamically added to the server. By default, the Ibis communication library comes with a registry service. This registry service manages pools, possibly multiple pools at the same time.

[1] `http://ant.apache.org`

In addition to the registry service, the server also allows Ibises to route traffic over the server if no direct connection is possible between two instances due to firewalls or NAT boxes. This is done using the Smartsockets library of the Ibis project.

The Ibis server is started with the `mpj-server` script which is located in the `bin` directory of the Ibis MPJ distribution. Before starting an Ibis MPJ application, an Ibis server needs to be running on a machine that is accessible from all nodes participating in the Ibis MPJ run. The server listens to a TCP port. The port number can be specified using the `--port` command line option to the `mpj-server` script. For a complete list of all options, use the `--help` option of the script. One useful option is the `--events` option, which makes the registry print out events.

3.2.1 Hubs

The Ibis server is a single point which needs to be reachable from every Ibis instance. Since sometimes this is not possible due to firewalls, additional *hubs* can be started to route traffic, creating a routing infrastructure for the Ibis MPJ instances. These hubs can be started by using mpj-server script with the `--hub-only` option. In addition, each hub needs to know the location of as many of the other hubs as possible. This information can be provided by using the `--hub-addresses` option. See the `--help` option of the mpj-server script for more information.

3.3 Running the example: preliminaries

When the Ibis server is running, the Ibis MPJ application itself can be started. There are a number of requirements that need to be met before Ibis (and thus Ibis MPJ) can be started correctly. In this section we will discuss these in detail.

Several of the steps below require the usage of *system properties*. System properties can be set in Java using the `-D` option of the `java` command. Be sure to use appropriate quoting for your command interpreter.

As an alternative to using system properties, it is also possible to use a java properties file [2]. A properties file is a file containing one property per line, usually of the format `property = value`. Properties of Ibis can be set in such a file as if they were set on the command line directly.

Ibis and Ibis MPJ will look for a file named `ibis.properties` in the current working directory, on the class path, and at a location specified with the `ibis.properties.file` system property.

3.3.1 Add jar files to the classpath

The Ibis MPJ implementation is provided in a single jar file: mpj.jar, appended with the version of Ibis MPJ, for instance `mpj-2.2.jar`. Ibis MPJ interfaces to Ibis using the Ibis Portability Layer, or *IPL*. Both Ibis MPJ and the IPL depend on various other libraries. All jar files in $MPJ_HOME/lib need to be on the classpath.

3.3.2 Configure Log4j

Ibis and Ibis MPJ use the Log4J library of the Apache project to print debugging information, warnings, and error messages. This library must be initialized. A configuration

[2] `http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html`

file can be specified using the `log4j.configuration` system property. For example, to use a file named `log4j.properties` in the current directory, use the following command line option: `-Dlog4j.configuration=file:log4j.properties`. For more info, see the log4j website [3].

3.3.3 Set the location of the server and hubs

To communicate with the registry service, each Ibis instance needs the address of the Ibis server. This address must be specified by using the `ibis.server.address` system property. The full address needed is printed on start up of the Ibis server.

For convenience, it is also possible to only provide an address, port number pair, e.g. `machine.domain.com:5435` or even simply a host, e.g. `localhost`. In this case, the default port number (8888) is implied. The port number provided must match the one given to the Ibis server with the `--port` option.

When additional hubs are started (see Section 3.2.1), their locations must be provided to the Ibis instances. This can be done using the `ibis.hub.addresses` property. Ibis expects a comma-separated list of addresses of hubs. Ibis will use the first reachable hub on the list. The address of the Ibis server is appended to this list automatically. Thus, by default, the Ibis server itself is used as the hub.

3.3.4 Set the name and size of the pool

Each Ibis instance belongs to a pool. The name of this pool must be provided using the `ibis.pool.name` property. With the help of the Ibis server, this name is then used to locate other Ibis instances which belong to the same pool. Since the Ibis server can service multiple pools simultaneously, each pool must have a unique name.

It is possible for pools to have a fixed size. In these so-called *closed world* pools, the number of Ibises in the pool is also needed to function correctly. This size must be set using the `ibis.pool.size` property. This property is normally not needed. When it is needed, but not provided, Ibis will print an error.

3.3.5 The mpj-run script

To simplify running an Ibis MPJ application, a `mpj-run` script is provided with the distribution. This script can be used as follows

`mpj-run` *java-flags class parameters*

The script performs the first two steps needed to run an Ibis MPJ application. It adds all required jar files to the class path, and configures log4j. It then runs `java` with any command line options given to it. Therefore, any additional options for Java, the main class and any application parameters must be provided as if `java` was called directly.

The `mpj-run` script needs the location of the Ibis MPJ distribution. This must be provided using the MPJ_HOME environment variable.

[3] `http://logging.apache.org/log4j`

3.4 Running the example on Unix-like systems

This section is specific for Unix-like systems. In particular, the commands presented are for a Bourne shell or bash.

We will now run the example. All code below assumes the MPJ_HOME environment variable is set to the location of the Ibis MPJ distribution.

First, we will need an Ibis server. Start a shell and run the `mpj-server` script:

```
$ $MPJ_HOME/bin/mpj-server --events
```

By providing the `--events` option the server prints information on when Ibis instances join and leave the pool.

Next, we will start the application two times. Run the following command in two different shells:

```
$ CLASSPATH=$MPJ_HOME/examples/lib/mpj-examples.jar \
    $MPJ_HOME/bin/mpj-run \
    -Dibis.server.address=localhost \
    -Dibis.pool.size=2 -Dibis.pool.name=test \
    sor.Sor 512
```

This sets the CLASSPATH environment variable to the jar file of the application, and calls mpj-run. You should now have two running instances of your application. One of them should print:

```
Running 512 x 512 SOR
SOR took 14.050 seconds
Used 1190 iterations, diff is 0.001645, allowed diff is 0.001646
```

or something similar.

As said, the mpj-run script is only provided for convenience. To run the application without mpj-run, the command below can be used. Note that this only works with Java 6. For Java 1.5, you need to explicitly add all jar files in $MPJ_HOME/lib to the classpath.

```
$ java \
    -cp \
    $MPJ_HOME/lib/'*':$MPJ_HOME/examples/lib/mpj-examples.jar \
    -Dibis.server.address=localhost \
    -Dibis.pool.name=test -Dibis.pool.size=2 \
    -Dlog4j.configuration=file:$MPJ_HOME/log4j.properties \
    sor.Sor 512
```

3.5 Running the example on Windows systems

We will now run the example on a Windows XP system. All code below assumes the MPJ_HOME environment variable is set to the location of the Ibis MPJ distribution.

To set environment variable on Windows, right-click on the 'My Computer' icon, 'Properties', 'Advanced' tab, 'Environment Variables' button. There, you can add variables to either the User variables or the System variables.

First, we will need an Ibis server. Start a command prompt window and run the `mpj-server` script:

```
C:\DOCUME~1\Temp> "%MPJ_HOME%"\bin\mpj-server --events
```

Note the quoting, which is needed when MPJ_HOME contains spaces.

By providing the `--events` option the server prints information on when Ibis instances join and leave the pool.

Next, we will start the application two times. Run the following commands in two different shells:

```
C:\DOCUME˜1\Temp> cd %MPJ_HOME%\examples
C:...> set CLASSPATH=lib\mpj-examples.jar
C:...> "%MPJ_HOME%"\bin\mpj-run
    "-Dibis.server.address=localhost"
    "-Dibis.pool.size=2" "-Dibis.pool.name=test"
    sor.Sor 512
```

This sets the CLASSPATH environment variable to the jar file of the application, and calls mpj-run. You should now have two running instances of your application. One of them should print:

```
Running 512 x 512 SOR
SOR took 14.050 seconds
Used 1190 iterations, diff is 0.001645, allowed diff is 0.001646
```

or something similar.

As said, the mpj-run script is only provided for convenience. To run the application without mpj-run, the commands below can be used. Note that this only works with Java 6. For Java 1.5, you need to explicitly add all jar files in %MPJ_HOME%\lib to the classpath.

```
C:\DOCUME˜1\Temp> cd %MPJ_HOME%\examples
C:...> java
    -cp "%MPJ_HOME%\lib\*";lib\mpj-examples.jar
    -Dibis.server.address=localhost
    -Dibis.pool.name=test -Dibis.pool.size=2
    -Dlog4j.configuration=file:"%MPJ_HOME%"\log4j.properties
    sor.Sor 512
```

4 Further Reading

The Ibis web page `http://www.cs.vu.nl/ibis` lists all the documentation and software available for Ibis, including papers, and slides of presentations.