Ibis Communication Library User's Guide

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

January 31, 2008

1 Introduction

This manual describes the steps required to run an application that uses the Ibis communication library. How to create such an application is described in the IPL Programmers manual.

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.

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.

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 `ibis-server` script which is located in the `bin` directory of the Ibis distribution. Before starting an Ibis application, an Ibis server needs to be running on a machine that is accessible from all nodes participating in the Ibis run. The server listens to a TCP port. The port number can be specified using the `--port` command line option to the `ibis-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 (such as Ibises joining a pool).

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 instances. These hubs can be started by using ibis-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 ibis-server script for more information.

3 Running an Ibis Application

When the Ibis server is running, the application itself can be started. There are a number of requirements that need to be met before Ibis 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.

As an alternative to using system properties, it is also possible to use a java properties file [1]. 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 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.

3.1 Add ipl.jar to the class path

An application interfaces to Ibis using the Ibis Portability Layer, or *IPL*. The code for this package is provided in a single jar file: ipl.jar, appended with the version of ibis, for instance `ipl-2.0.jar`. This jar file needs to be added to the class path of the application.

3.2 Provide the Ibis implementations

The IPL loads the actual Ibis implementation dynamically. These implementations (and their dependencies) can be provided in two ways:

1. Add the jar files of the implementations and their dependencies to the class path
2. Set the `ibis.implementation.path` system property to the location of the Ibis implementations and dependencies.

The `ibis.implementation.path` property is a list of directories, separated by the default path separator of your operating system. In Unix, this is the `:` character, in Windows it is a `;`.

3.3 Configure Log4j

Ibis uses the Log4J library of the Apache project to print debugging information, warnings, and error messages. This library must be initialized. A configuration 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 [2].

[1] `http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html`
[2] `http://logging.apache.org/log4j`

3.4 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 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.

3.5 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.

4 The ibis-run script

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

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

The script performs the first three steps needed to run an application using Ibis. It adds the ipl.jar and all Ibis implementation jars 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 `ibis-run` script needs the location of the Ibis distribution. This must be provided using the IBIS_HOME environment variable.

5 Example

To illustrate running an Ibis application we will use a simple ”Hello World” application. This application is started twice on a single machine. One instance will send a small message to the other, which will print it.

5.1 Compiling the example

The example applications for the Ibis communication library are provided with the Ibis distribution, in the `examples` directory. For convenience, these applications are already compiled.

If you change any of the example, you will need to recompile them. This requires the build system `ant`[3]. Running `ant` in the examples directory compiles the examples.

Alternatively, they can be compiled using only `javac`. The sources are located in the `src` directory of examples. Be sure to add `ipl.jar` from the `lib` directory of the distribution to the class path.

5.2 Running the example

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

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

```
$ $IBIS_HOME/bin/ibis-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. One instance will act as the ”server”, and one the ”client”. The application will determine who is who automatically. Therefore we can using the same command line for both client and server. Run the following command in two different shells:

```
$ CLASSPATH=$IBIS_HOME/examples/lib/ipl-examples-2.0.jar \
    $IBIS_HOME/bin/ibis-run \
    -Dibis.server.address=localhost -Dibis.pool.name=test \
    ibis.ipl.examples.Hello
```

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

```
 Server received: Hi there
```

As said, the ibis-run script is only provided for convenience. To run the application without ibis-run, the following command can be used:

```
$ java \
    -cp \
    $IBIS_HOME/lib/ipl-2.0.jar:$IBIS_HOME/examples/lib/ipl-examples-2.0.jar \
    -Dibis.impl.path=$IBIS_HOME/lib \
    -Dibis.server.address=localhost \
    -Dibis.pool.name=test \
    -Dlog4j.configuration=file:$IBIS_HOME/log4j.properties \
    ibis.ipl.examples.Hello
```

In this case, we use the `ibis.impl.path` property to supply Ibis with the jar files of the Ibis implementations. Alternatively, they could also all be added to the class path.

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

6 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.

For detailed information on developing an Ibis application see the Programmers Manual, available in the docs directory of the Ibis distribution.