Satin Divide-and-Conquer System User's Guide

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

December 9, 2009

1 Introduction

This manual describes the steps required to run an application that uses the Satin Divide-and-Conquer System. How to create such an application is described in the Satin Divide-and-Conquer Programmer's Manual. Here, we will discuss how to compile and run the example that is described in the Programmer's Manual.

Since Satin is built on top of the Ibis Portability Layer (IPL), the Satin release contains the Ibis communication library, which contains implementations of the IPL. Parts of this manual may look familiar for readers that are familiar with the Ibis communication library.

2 Compiling the Fibonacci example

The Fibonacci example application for the Satin Divide-and-Conquer System is provided with the Satin distribution, in the `examples` directory. For convenience, the application is 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 example, and rewrites the class files for use with Satin.

Invoking *ant clean compile* compiles a sequential version of the application, leaving the class files in a directory called `tmp`.

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 first compile your application to class files or jar files, and then process those using the *satinc* script. This script can be found in the Satin bin directory. It takes either directories, class files, or jar files as parameter, and processes those, possibly rewriting them for Ibis. In case of a directory, all class files and jar files in that directory or its subdirectories are processed. The command sequence

```
$ cd $SATIN_HOME/examples
$ mkdir tmp
$ javac -d tmp -g \
     -classpath ../lib/satin-2.2.jar \
     src/fib/*.java
$ CLASSPATH=tmp ../bin/satinc -satin "fib.Fib" tmp
$ mkdir lib
```

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

```
$ ( cd tmp ; jar c . ) > lib/satin-examples.jar
$ rm -rf tmp
```

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

Note that the Satin bytecode rewriter `satinc` needs to know the classes in your application that must be rewritten for Satin. Those classes are: the main class, any class that implements a shared object, and all classes that invoke spawns or syncs. When using the `satinc` script, these classes must be provided as a comma-separated list, with the `-satin` option. In `build.xml` they are provided as the `satin-classes` property.

The satinc script can also be used to give advice on locations of sync method invocations. The examples include a special version of Fib to demonstrate this feature. For instance after calling javac, as above, you can call

```
$ CLASSPATH=tmp ../bin/satinc -syncadvise "fib.FibWithoutSync" tmp
```

which will print suggested locations for sync method invocations on standard output. Note that the program ignores sync invocations that are already present, and also that it is not fullproof: in obscure cases it is too conservative, which means that your application will behave like a sequential program. If you are satisfied that it gives proper locations, you can have them inserted in the bytecode automatically, by replacing the satin bytecode rewriter line with:

```
$ CLASSPATH=tmp ../bin/satinc -syncrewriter "fib.FibWithoutSync" -satin "fib.Fi
```

3 A Satin run

Before discussing the running of a Satin 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.

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

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

3.3 Running the example: preliminaries

When the Ibis server is running, the Satin application itself can be started. There are a number of requirements that need to be met before Ibis (and thus Satin) 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 Satin 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 Satin implementation is provided in a single jar file: satin.jar, appended with the version of Satin, for instance `satin-2.2.jar`. Satin interfaces to Ibis using the Ibis Portability Layer, or *IPL*. Both Satin and the IPL depend on various other libraries. All jar files in $SATIN_HOME/lib need to be on the classpath.

3.3.2 Configure Log4j

Ibis and Satin use 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 follow-

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

ing 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 Satin system properties

Satin recognizes the following system properties, which can either be provided on the command line, or in a `ibis.properties` file as discussed above:

`satin.closed` Only use the initial set of hosts for the computation; do not allow further hosts to join the computation later on. This requires the `ibis.pool.size` property as well, as discussed above.

`satin.stats` Display some statistics at the end of the Satin run. This is the default.

`satin.stats=false` Don't display statistics.

`satin.detailedStats` Display detailed statistics for every member at the end of the Satin run.

`satin.alg`=*algorithm* Specify the load-balancing algorithm to use. The possible values for *algorithm* are: RS for random work-stealing, CRS for cluster-aware random-work stealing, and MW for master-worker.

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

`satin.ft.naive` Fault tolerance in Satin is implemented by recomputing jobs lost as a result of processor crashes. To improve the performance of the algorithm, a Global Result Table (GRT) may be used, in which results of so-called ”orphan” jobs are stored, which otherwise would have to be discarded. The results of orphan jobs are reused while recomputing jobs lost in crashes. The `satin.ft.naive` property specifies that Satin is to use the naive version of its fault tolerance implementation, without a GRT. This means that sub-jobs of lost jobs will all be recomputed. The default is that a GRT is used.

3.3.6 The satin-run script

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

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

The script performs the first two steps needed to run an application using Satin. 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 `satin-run` script needs the location of the Satin distribution. This must be provided using the SATIN_HOME environment variable.

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 SATIN_HOME environment variable is set to the location of the Satin distribution.

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

```
$ $SATIN_HOME/bin/satin-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 ”Satin master”, the other one will be a ”Satin client”. Satin will determine who is who automatically. Therefore we can using the same command line for both master and client. Run the following command in two different shells:

```
$ CLASSPATH=$SATIN_HOME/examples/lib/satin-examples.jar \
    $SATIN_HOME/bin/satin-run \
    -Dsatin.closed -Dibis.server.address=localhost \
    -Dibis.pool.size=2 -Dibis.pool.name=test \
    fib.Fib 32
```

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

```
Running Fib 32
application time fib (32) took 10.787 s
application result fib (32) = 2178309
```

or something similar, followed by some Satin statistics.

As said, the satin-run script is only provided for convenience. To run the application without satin-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 $SATIN_HOME/lib to the classpath.

```
$ java \
    -cp \
    $SATIN_HOME/lib/'*':$SATIN_HOME/examples/lib/satin-examples.jar \
    -Dibis.server.address=localhost \
    -Dibis.pool.name=test -Dibis.pool.size=2 \
    -Dsatin.closed \
    -Dlog4j.configuration=file:$SATIN_HOME/log4j.properties \
    fib.Fib 32
```

3.5 Running the example on Windows systems

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

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

```
C:\DOCUME~1\Temp> "%SATIN_HOME%"\bin\satin-server --events
```

Note the quoting, which is needed when SATIN_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. One instance will act as the ”Satin master”, the other one will be a ”Satin client”. Satin will determine who is who automatically. Therefore we can using the same command line for both master and client. Run the following command in two different shells:

```
C:\DOCUME~1\Temp> set CLASSPATH=%SATIN_HOME%\examples\lib\satin-examples.jar
C:\DOCUME~1\Temp> "%SATIN_HOME%"\bin\satin-run
    "-Dsatin.closed" "-Dibis.server.address=localhost"
    "-Dibis.pool.size=2" "-Dibis.pool.name=test"
    fib.Fib 32
```

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

```
Running Fib 32
application time fib (32) took 10.787 s
application result fib (32) = 2178309
```

or something similar, followed by some Satin statistics.

As said, the satin-run script is only provided for convenience. To run the application without satin-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 %SATIN_HOME%\lib to the classpath.

```
C:\DOCUME~1\Temp> java
    -cp "%SATIN_HOME%\lib\*";"%SATIN_HOME%"\examples\lib\satin-examples.jar
    -Dibis.server.address=localhost
    -Dibis.pool.name=test -Dibis.pool.size=2
    -Dsatin.closed
    -Dlog4j.configuration=file:"%SATIN_HOME%"\log4j.properties
    fib.Fib 32
```

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.

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