

Wikiprint Book

Title: net-iocache

Subject: Java Path Finder - projects/net-iocache

Version: 17

Date: 03/06/2013 12:34:02 AM

Table of Contents

net-iocache	3
What is net-iocache?	3
How to download.	3
How to build.	3
Installation	3
Running net-iocache.	4
Running the examples.	4
Other options/tuning.	5
Jar archives	5

net-iocache

A JPF extension for model checking networked programs, by Watcharin Leungwattanakit and Cyrille Artho.

For any information or to report problems, please contact:

Watcharin Leungwattanakit: watcharin@...

Note: The documentation below refers to the new version, to be used with the current distribution of Java PathFinder from this page. To use a pre-built release, which works against JPF v4 from <http://javapathfinder.sourceforge.net>, go to the archive page containing the previous version: [projects/net-iocache-v4](http://projects.net-iocache-v4)

What is net-iocache?

Net-iocache is a Java PathFinder extension for verifying networked applications. JPF executes one of the processes in an application, whereas the others run on the standard Java virtual machine. Net-iocache captures inputs/outputs of the target program and replays them when the program needs again after backtracking. The algorithm to be run in net-iocache is presented in

- C. Artho, W. Leungwattanakit, M. Hagiya, Y. Tanabe, M. Yamamoto. Cache-Based Model Checking of Networked Applications: From Linear to Branching Time. ASE 2009, November 2009, Auckland, New Zealand.
- C. Artho, W. Leungwattanakit, M. Hagiya, and Y. Tanabe. Efficient model checking of networked applications. In Proc. TOOLS EUROPE 2008, volume 19 of LNBIIP, pages 22–40, Zurich, Switzerland, 2008. Springer.

Currently, programs are assumed to be either running as a server, accepting connections from several clients, or as a client, connecting to one server. The algorithm has been tested for depth-first search. Other search types, and other types of protocols (such as peer-to-peer) are not supported yet.

How to download.

You can obtain the source code of the extension using Mercurial (hg):

```
hg clone http://babelfish.arc.nasa.gov/hg/jpf/jpf-net-iocache
```

The new version comes with a `jpf.properties` file for configuration with the new version of JPF, and automatically generates a `~/jpf/site.properties` file when the project is built with Apache Ant. The generated file should work if you have checked out `jpf-net-iocache` as a subdirectory of the overall `jpf` repository, with `jpf-core` being another subdirectory. For example, JPF resides in `project/jpf-core`, the extension in `project/jpf-net-iocache`. Please refer to the README file for details.

How to build.

You can either build Java PathFinder from the command line with Ant, or from within Eclipse. To compile the `jpf-net-iocache` project, we recommend building sources with Apache Ant since it will automatically check and generate file `site.properties` if necessary.

To build with Ant, switch to the directory where the `jpf-net-iocache` extension is located (where this file is located), and run

```
ant
```

which should compile all `jpf-net-iocache` sources including example programs.

Installation

When you have built the `jpf-net-iocache` project, three jar files are created under directory `build`:

- `jpf-net-iocache.jar`: The implementation of the cache.
- `jpf-net-iocache-classes.jar`: The model classes.
- `jpf-net-io-cache-examples.jar`: Examples.

You can run net-iocache without manual installation. All jar files should be under directory `build` if you build the project with Apache Ant.

The archive names refer to the new naming convention with JPF 5 and are easily distinguished from the distinct file names for the extension that works with JPF 4 (see [projects/net-iocache-v4](http://projects.net-iocache-v4)).

Running net-iocache.

If you want to run your own program, you should set `JPF_CORE` to the directory you installed the `jpf-core` project. You then need to specify the following options to JPF to use `net-iocache`:

```
java -jar ${JPF_CORE}/build/RunJPF.jar \  
+classpath="[classpath to your application]" \  
+sourcepath="[source path to your application]" \  
+report.console.property_violation=error,trace \  
+listener=gov.nasa.jpf.network.listener.CacheNotifier,gov.nasa.jpf.network.listener.CacheLogger \  
+jpf-net-iocache.boot.peer.command="[for server:command to start client process]" \  
+jpf-net-iocache.arg0="[argument 0 of the client process]" \  
+jpf-net-iocache.arg1="[argument 1 of the client process]" \  
...  
Application [application_args]
```

The first argument ensures that JPF can find your application classes. The second argument (`sourcepath`) points to the source code of your application. If JPF finds an error in your application, the error trace with source code will be displayed. The third argument (`property_violation`) tells JPF to report the trace to the error it has detected; the fourth argument (`listener`) activates the cache. If a client is verified, `jpf-net-iocache.boot.peer.command` can be omitted; if a server is verified, the cache needs to know what kind of process to spawn when the server is waiting for a client to connect. You can supply arguments for the client process by adding option `jpf-net-iocache.arg[n]` where `n` is the `n`th argument.

An alternative way to execute JPF with several options is to create an application property file. From the sample command above, you can create property file like this:

```
target = [Application]  
target_args = [application_args]  
classpath = [classpath to your application]  
sourcepath = [source path to your application]  
report.console.property_violation = error,trace  
listener = gov.nasa.jpf.network.listener.CacheNotifier,gov.nasa.jpf.network.listener.CacheLogger  
jpf-net-iocache.boot.peer.command = [for server:command to start client process]  
jpf-net-iocache.arg0 = [argument 0 of the client process]  
jpf-net-iocache.arg1 = [argument 1 of the client process]  
...
```

Suppose that you save the above property file as `myapplication.jpf`, the command starting JPF becomes like this:

```
java -jar ${JPF_CORE}/build/RunJPF.jar myapplication.jpf
```

For other (optional) arguments to `net-iocache`, see below ("tuning"). For sample start-up scripts, please see the scripts ending with `-mc.sh` in directory `bin`.

This project comes with a default property file, `jpf.properties`. It contains default JPF options whose namespace is `"jpf-net-iocache"`. The default options usually need not be modified. If you want to specify additional options for your program, please add them into your application property file or the command line starting JPF. Please see how JPF properties are applied from <http://babelfish.arc.nasa.gov/trac/jpf/wiki/user/config>.

Running the examples.

You can find the scripts to run the example programs in directory `"bin"`. You may want to make sure that `net-iocache` is properly built by running

```
./alphabet-client-mc.sh 2 1
```

which verifies the alphabet client with two threads inside JPF and generates the alphabet server process to handle the client threads.

The scripts that end with `"-mc"` start JPF verifying an example program and the peer process of the program.

You can write your own script by using one of the provided scripts as a template. All `"mc"` scripts include file `"env.sh"`, which initializes shell variables used in most scripts to default values. The meaning of each variable is described below.

- `JPF_HOME`: JPF base directory (default: the parent directory of `jpf-net-iocache`)

- `JPF_CORE`: jpf-core base directory (default: directory `jpf-core` under `JPF_HOME`)
- `SRC_DIR`: jpf-net-iocache base directory (default: directory `jpf-net-iocache` under `JPF_HOME`)
- `LIB_DIR`: directory containing necessary jar files
- `BIN_DIR`: directory containing the startup scripts
- `VM_ARG`: This contains arguments for the Java virtual machine.
- `CLASSPATH`: Classpaths that JPF needs including JPF core classes, model classes, native peer classes, and example program classes.
- `RUN_EXAMPLE_CMD`: A command to start a JPF process verifying an example program. It includes default options as shown in section [Running net-iocache](#). You can change values and add extra options by adding "+<option>=<value>" to this variable.
- `OPTION_PREFIX`: The prefix of options specific to jpf-net-iocache. (default: `jpf-net-iocache`)

Other options/tuning.

net-iocache introduces three new options: `exception.simulation`, `main.termination`, and `lazy.connect`.

- `exception.simulation`: If this option is set to true, calls to a network operation will result in either success or failure. Both possibilities are simulated by JPF. (Default: false)
- `main.termination`: If it is true, the main thread will run solo until the end before the other threads start running. This option would improve performance of verification when the main thread does nothing but creating worker threads. (Default: true)
- `lazy.connect`: If it is true, net-iocache will not open a physical connection immediately after the program calls method "connect". (Default: true)

Jar archives

The jar files at the bottom of this page contain builds against the old version of JPF (version 4) from sourceforge.net. Archives for v5 are coming out soon.