SVMDCP Net Layers Test Cases

December 15, 2003

Contents

1 General Description

The test cases are in the `test/` subfolder of the NetLayers package. They are used to test the functionality of the package, as well as the performance of different network technologies.

These test cases are separated into 2 groups: synchronous calls and asynchronous calls. For each test, a server is first startup, which prints its ID to the screen and also stores the ID in a file called `server.id` in the local directory. A client is started a few second after this. The client reads the server ID from the file and connects to it. Once the connection is set up, the client calls the function, a *copy function* that simply returns the string parameter, of the server 1000 times. After this, the client prints a message to the screen about the time that has been spent on the calls. The time for setting up the connection is not counted. Both the client and the server halt afterward.

These test cases are run locally on a single machine. The communication is between 2 processes.

2 Test Requirements

In each of directories under `test/`, there are two scripts:

- `localtest_ss`, synchronous test. This script runs the server that provides the copy function and the synchronous client.
- `localtest_as`, asynchronous test. This script runs the same copy server and the asynchronous client.

Prior to running these test scripts, please make sure the package is correctly made, and the following environment variables are set:

- `LD_LIBRARY_PATH`. It must contain the path where the library of MPI interface can be found. This library is located at the `mpi/` subfolder of the net layers package. This environment variable is needed to run the MPI test cases.
- `PVM_ROOT` and `PVM_ARCH`, the path where PVM is installed and the architecture of your machine. For RedHat Linux on a x86 machine, the `PVM_ARCH` is usually `LINUXI386`; for SuSE Linux, it is usually `LINUX`. This architecture string can be found by looking at the name of the subfolder in `$PVM_ROOT/lib/`. These environment variables are necessary to compile JavaPVM.
- `JAVAPVM_PATH`, the path where JavaPVM is installed. This environment variable is only necessary for the JavaPVM test cases.
- `JPVM_PATH`, the path where JPVM is installed. This environment variable is only necessary for the JPVM test cases.
- `PATH`. Make sure the following programs are in your `$PATH`: `java`, `javac`, `orbd`, `rmiregistry`, `pvm`, `lamboot`, `lamhalt` and `mpirun`.

Note that to compile JavaPVM in most Linux distributions such as RedHat and SuSE, a patch in the `test/JavaPVM/` subfolder of the NetLayers package is needed. Download the source package from the JavaPVM website, unpack it, `cd` to the folder, and apply the patch with this command:

```
patch -p0 < jPVM-v1.1.4.patch
```

3 Server-Client Model

As mentioned, the server provides a copy function which can be expressed as:

```
String copy(String s)
```

Where the return value is always equal to the parameter `s`. This server is first started. It can use any of the implemented protocol as the underlying network technology that supports its service. In these test cases, the servers are the same program. The protocol that it uses completely depends on the parameter given to it. The following statement is to create the corresponding network object:

```
Network net = (Network) Class.forName("svmdcp.NetLayers." + argv[0]).newInstance();
```

After the network is initialized, the server reveals its ID to the client. For example, if RMI is the chosen protocol, its ID may look like `rmi://localhost:1099/MessageReceiver.R1234208140`. This means the network protocol is `RMI`, the server is located on host `localhost` and port `1099` (default port number for RMI), and its unique ID on this host is `MessageReceiver.R1234208140`.

The client is started later. It uses this ID string to detect the protocol that the server uses and creates a network object that is compatible with the one on the server. For example, if the user uses RMI as its network protocol, the client must also use RMI. This compatible network protocol allows the client to connect to the server. After a connection is set up, calls can be made.

There are 2 kinds of clients: asynchronous client that calls asynchronously and synchronous client. Because they automatically detects the correct protocol to be used, they can communicate with servers that use different protocols.

4 Threading Problems and Other Known Problems

As JavaPVM and MPI network implementation requires linking Java code with native C libraries, there are threading problems in the asynchronous test cases of them. As a result, asynchronous calls are highly discouraged for these two protocols.

Currently there is no thread-safe MPI implementation yet. Because of this, the class that interfaces MPI with the NetLayers package uses a separate thread to handle all the calls to the MPI library. This works find for synchronous calls, though the performance is greatly compromised. However, MPI is dangerous for asynchronous calls, mainly due to its lack of buffering space for incoming messages and outgoing messages. In the test case that uses MPI as the protocol that supports asynchronous calls, 1000 calls are sent to the server asynchronously, without waiting for the response of the server. This usually exhausts the very very limited buffer space on both the client side and the server side. As a result, both of them hangs or segment faults occur. This tester and user of the package must be aware of this fact.