



IP-FP6-015964

AEOLUS

Algorithmic Principles for Building Efficient Overlay Computers

Deliverable D6.2.2

Microbenchmarking Software Package

Paolo Bertasi, Mauro Bianco, Giorgio Ollearo, Andrea Pietracaprina, Geppino Pucci
Dipartimento di Ingegneria dell'Informazione, Università di Padova, Padova, Italy

Responsible Partner: Università degli Studi di Padova (UNIPD)
Report Preparation Date: March 2007

Contract Start Date: 01/09/05 Duration: 48 months
Project Co-ordinator: University of Patras (EL)

Contents

1	Introduction	1
2	Implementation choices	2
3	The library APIs	3
3.1	Slave mode	3
3.2	Master mode	3
3.2.1	CPU test	4
3.2.2	Point-to-point test	5
3.2.3	Scatter test	5
3.2.4	Gather test	6
3.2.5	All-to-all test	6
4	Installation	7
4.1	Command line guide	8
5	Additional Documentation	8

1 Introduction

This microbenchmarking suite, dubbed **SuiteNG**, is intended to collect statistics and performance measures on JXTA peers. Specifically, the library implementing the suite provides essentially measures of bandwidth among chosen peer configurations and measures of computational power of individual peers. All the tests are carried out in a client/server fashion. Each peer can act as a client querying the other peers or as a server offering its “testability”. To underline this difference with the standard client/server paradigm we call *master* the querying peer and *slave* the peer to be tested.

The suite has been tested in a highly heterogeneous environment. Indeed, our testbed consists of a number of computers with very different capabilities and architectures linked by high-speed connections. Namely, the machines are linked either through a 100 Mb switched Ethernet LAN or with broadband connections (see Figure 1).

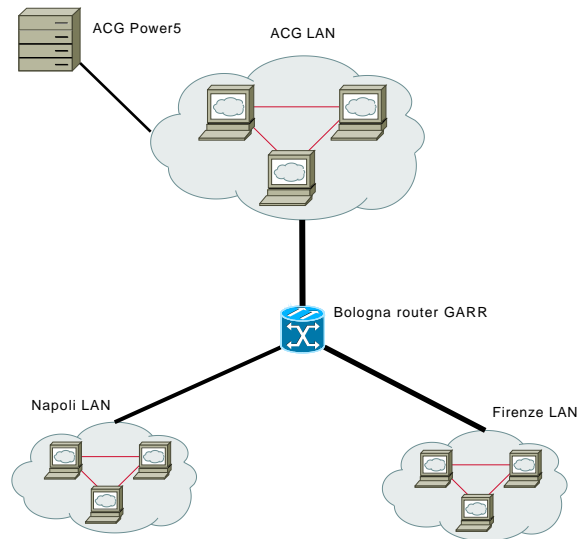


Figure 1: The networked testbed used for exercising the microbenchmarking suite

2 Implementation choices

As mentioned above, the core of the suite is the library that provides the functionality to test a peer and to allow a peer to be tested. The main class is a parser, called `Test.java`, which invokes the right library method to start the peer as:

- master peer
- slave peer
- rendezvous peer
- relay peer

The latter two methods are not essential to run the test. However they are very useful in a complex environment managed by firewall and/or NAT.

We decided to let the network start in the main class and then call the library to perform the test.

3 The library APIs

The main class of the library is the class `BenchmarkSuite`. Its methods are called to use the peer as slave peer or as master and to choose the test to perform. The public methods are listed below.

`beTestable` transforms the peer in a slave peer accepting test requests.

`beNotTestable` stops the possibility to test the peer.

`doCpuTest` discovers the requested number of slaves and performs a CPU test.

`doBandwidthTest` discovers the requested number of slaves and performs a point-to-point bandwidth test with each slave sequentially. This method can also be used also to get latency measures.

`doGatherTest` discovers the requested number of slaves and performs a gather test (all-to-one communication towards the master peer).

`doScatterTest` discovers the requested number of slaves and performs a scatter test (one-to-all communication towards the slave peers).

`doAll2AllTest` discovers the requested number of slaves and performs an all-to-all test.

3.1 Slave mode

When running in slave mode, the peer accepts the request originating from a master. The peer can perform more than one test at a while. However, the measures collected by a test while other tests are running may turn out to be rather inaccurate. To avoid this problem we let the user choose a period of time in which the slave cannot accept further tests after successfully accepting one.

Neither `beTestable()` nor `beNotTestable()` has arguments. The first method switches the peer in slave accepting mode and the second reverts the peer to a normal peer not accepting test request.

3.2 Master mode

All the following methods, before starting the real test, try to discover the needed amount of slave peers. They also use the class called `Communicator` to manage the communication over the JXTA socket. This class was written to overcome the well know unreliability of the `jxtaSocket` implementation.

In the suite there are two kinds of time limit. The first is a hard limit defined by the settings of the communication class, called `Communicator`. If the elapsed time between two messages is greater than this limit the communication is restarted and the test is

considered failed. The default value is 102 seconds for the initiator of the communication and 215 seconds for the other peer. The second limit is the limit on the duration of the selected test. The master, after this period, usually set to 3000 seconds stops its test class, if the test is not finished.

Here is a brief but complete review of the APIs of the master mode.

3.2.1 CPU test

This test uses a quiz-like approach to investigate the computational power available on the inquired slaves. The master asks the peer to perform a FMA¹ on three randomly initialized arrays. The test can be performed fixing the amount of time to be spent in the experiment or fixing the number of iterations requested. The computations can be done using *integer* or *floating point* numbers.

In a peer-to-peer environment, it is important to deal with the risk of cheating peers. So in our test we have provided the possibility to check the correctness of the results processed by the inquired peer.

The results are expressed in iterations per seconds.

Syntax

```
doCpuTest(int numRepetitions, int numRequestedPeers, boolean isImposingTime,
int timeOrCycles, int initialSeed, long maxTime, boolean floatingPoint, boolean
trusted)
```

`int numRepetitions` number of repetitions to be performed.

`int numRequestedPeers` number of slave peers to be involved.

`boolean isImposingTime` `true` if the test has to be performed by fixing the amount of time. `false` if the duration of the test is bound by the number of iterations required.

`int timeOrCycles` the number of milliseconds or iterations requested depending on the previous boolean.

`int initialSeed` the value of the initial seed used to generate the pseudo-random numbers needed to initialize the arrays.

`long maxTime` the maximum number of milliseconds allowed for the test execution.

`boolean floatingPoint` `true` if the test has to be performed using floating point numbers, `false` if it uses integer.

¹The acronym FMA refers to the Fused Multiply-and-Add operation, that is, a floating-point multiply-add operation performed in one step. Our test performs the following operation: `c[i] += a[i]*b[i];`

boolean `trusted` it describes the trustfulness of the environment. `true` if the slaves are assumed to be honest. In this case the time measures are taken by the slaves themselves. `false` if the behavior of the slaves cannot be trusted. The time measures are taken by the master and the final results are re-calculated by the master to detect any cheat.

3.2.2 Point-to-point test

This test measures the “quality” of a point to point unidirectional transmission between two peers. The master peer sends an arbitrary amount of raw data to the target peer and measures the time elapsed. Through this test it is possible to obtain bandwidth and latency measures. The test makes possible to perform multiple sequential tests using the same amount of data or using exponentially increasing amounts of data. When increasing the data, the packet size double at every iteration.

Syntax

```
doBandwidthTest(int numRepetitions, int numRequestedPeers, int dataSize, boolean isDynamic, long maxTime)
```

`int numRepetitions` number of repetitions to be performed.

`int numRequestedPeers` number of slave peers to be involved.

`int dataSize` the amount of data to be sent; if the test is with increasing size this integer describes the size of the initial data chunk.

boolean `isDynamic` `true` if the peers double the dimension of the transmitted data chunk after each repetition.

`long maxTime` the maximum number of milliseconds allowed for the test execution.

3.2.3 Scatter test

The purpose of this test is to measure the bandwidth used by JXTA during a scatter operation. The master peer contacts a number of other peers and simultaneously starts to send data to the peers. As for the previous test, scatter can perform multiple sequential tests using same amount of data or using increasingly larger amounts of data.

Syntax

```
doScatterTest(int numRepetitions, int numRequestedPeers, int dataSize, boolean isDynamic, long maxTime)
```

`int numRepetitions` number of repetitions to be performed.

`int numRequestedPeers` number of slaves peer to be involved.

`int dataSize` the amount of data to be sent; if the test is with increasing size this integer describes the dimension of the initial data chunk sent to each slave.

`boolean isDynamic true` if the peers double the dimension of the transmitted data chunk after each repetition.

`long maxTime` the maximum number of milliseconds allowed for the test execution.

3.2.4 Gather test

The purpose of this test is to measure the bandwidth used by JXTA during a gather operation. The master peer contacts a number of other peers and tries to synchronize them. Here the synchronization steps performed by the master peer:

1. send three pings to the peers.
2. calculate the average (avg) of the three pings for each peers.
3. delay the start signal, for the successive operation, by “avg/2” for each peer.

The peers start to send the data to the master peer roughly at the same time regardless of their distance. As for the other tests, scatter can perform multiple sequential tests using the same amount of data or exponentially increasingly larger amounts of data.

Syntax

```
doGatherTest(int numRepetitions, int numRequestedPeers, int dataSize, boolean isDynamic, long maxTime)
```

`int numRepetitions` number of repetitions to be performed.

`int numRequestedPeers` number of slaves peer to be involved.

`int dataSize` the amount of data to be sent; if the test is with increasing size this integer describes the dimension of the initial data chunk sent to each slave.

`boolean isDynamic true` if the peers double the dimension of the transmitted data chunk after each repetition.

`long maxTime` the maximum number of milliseconds allowed for the test execution.

3.2.5 All-to-all test

The purpose of this test is to measure the bandwidth used by JXTA during a transmission from all peers to all peers. The master peer participates to the data exchange but also generates the information to create the connections and collects the results.

Syntax

```
doAll2AllTest(int numRepetitions, int numRequestedPeers, int dataSize, boolean  
isDynamic, long maxTime)
```

`int numRepetitions` number of repetitions to be performed.

`int numRequestedPeers` number of slaves peer to be involved.

`int dataSize` the amount of data to be sent; if the test is with increasing size this integer describes the dimension of the initial data chunk sent to each slave.

`boolean isDynamic` true if the peers double the dimension of the transmitted data chunk after each repetition.

`long maxTime` the maximum number of milliseconds allowed for the test execution.

4 Installation

In order to have a microbenchmarking suite running the jar file has to be copied onto every peer. Then the suite can be invoked as a slave or as a master requesting a peer. (The suite can also be invoked to establish either a relay or a rendezvous peer.) The following is a quick step-by-step guide on how to install the suite on UNIX/Linux machines.

1. Copy the suite to the target machines, e.g.

```
scp Benchmarking.jar username@machine.dei.unipd.it:~
```

2. copy the JXTA and AEOLUS library to the target machines, e.g.

```
scp -r jxtalib_directory username@machine.dei.unipd.it:~
```

3. connect to the selected machines to start the slave peers, e.g.

```
ssh username@machine.dei.unipd.it  
java -jar Benchmarking.jar edge
```

4. connect to the master peer to start the test, e.g.

```
java -jar Benchmarking.jar ma cpu 6 10 1000 cycles 1000000 15979
```

5. wait for the results

By default, the suite loads the list of rendezvous and relay peers from `http://verona.dei.unipd.it/~jxta`

These are the current lists in the files:

`rdvs.txt`:

`tcp://147.162.96.109:9701`

`relay.txt`:

`tcp://147.162.96.121:9701`

4.1 Command line guide

These are the self explanatory available command lines associated to the services offered by the suite. The prefix of every line is clearly:

```
java -jar Benchmarkink.jar ma
```

```
cpu NUM_PEERS NUM_REP MAX_TIME IS_TIME_OR_CYCLES TIME_OR_CYCLES
```

```
START_VALUE IS_FLOATING IS_TRUSTED
```

```
bandwidth NUM_PEERS NUM_REP MAX_TIME DATA_SIZE DYNAMICITY
```

```
scatter NUM_PEERS NUM_REP MAX_TIME DATA_SIZE DYNAMICITY
```

```
gather NUM_PEERS NUM_REP MAX_TIME DATA_SIZE DYNAMICITY
```

```
all2all NUM_PEERS NUM_REP MAX_TIME DATA_SIZE DYNAMICITY
```

5 Additional Documentation

For an extensive description of the design choices behind the chosen microbenchmarking approach, see [1]. Preliminary experiments and evaluation of the results have already been published in the open literature [2, 3].

References

- [1] M.Bianco, P. Bertasi, A. Pietracaprina, and G. Pucci. Microbenchmarking Software Package: Design Report. *AEOLUS Deliverable 3.2.2*, August 2006.

- [2] M.Bianco, P. Bertasi, A. Pietracaprina, and G. Pucci. Obtaining Performance Measures through Microbenchmarking in a Peer-to-Peer Overlay Computer. In *Proc. 1st International Workshop on P2P, Parallel, Grid and Internet Computing, 3PGIC 2007*, pages 285-290, Vienna, A, March 2007.
- [3] M.Bianco, P. Bertasi, A. Pietracaprina, and G. Pucci. Obtaining Performance Measures through Microbenchmarking in a Peer-to-Peer Overlay Computer. *International Journal of Computational Intelligence Research*. Accepted for publication, July 2007. To appear.