Jac-a-boo
JAva Cluster Application BOOtstrapper

Luc Hogie (I3S (**CNRS**, UNS), INRIA)

November 12, 2014

Abstract

Jac-a-boo is a middleware enabling the execution of Java applications across clusters. As such, it addresses the deployment of JVMs and classes, the communications infrastructure and the programming model.

The design objectives of Jac-a-boo is to be make distribution and parallelism as much transparent and easy as possible.

Contents

1 Introduction

We need a middleware for the remote execution of Java code. Take this code, bundled with this data, and run it on that computer.

There exist a number of existing middlewares for the distribution and parallelization of Java applications. But only ??? provide automatic deployment. Only ??? supports UDP

RMI (Remote Method Invocation) is the obvious solution to distributing code in Java, as it comes with the standard edition of Java. As such it has excellent integration with the Java platform. It is a mature solution now offering good performance. Unfortunately has no deployment mechanism and works on TCP only.

ProActive is Java grid middleware for parallel, distributed, and multi-threaded computing. It is developed by the OW2 Consortium, including INRIA, CNRS, University of Nice Sophia Antipolis, and the ActiveEon company. Pros:

- we may have local support
- provides access to clusters

Cons:

- no deployment mechanism.
- TCP-only
- not targeted to performance
- cumbersome
- "active object" distribution-model does not suit our needs

JPPF http://www.jppf.org/ No transparents management of results and exceptions (need to catch them and assign them for returning, the application then get it via myTask.getException();)

Hazelcast executor service http://hazelcast.org/docs/3.1/manual/html-single/#ExecutorService

Korus http://code.google.com/p/korus/ is thread-oriented. Need manual deployment/execution.

Java implementations of MPI (like G-JAVAMPI, JMPI, mpiJava, etc) can be used when message-passing is required. But MPI only support TCP as its transport protocol and it does not solve the issue of deployment and execution. It does not support data compression. Pros:

- major distribution platform
- excellent performance

Cons:

- no deployment mechanism.
- message-only
- TCP-only

CAJO

JGroups http://www.jgroups.org/ JGroups is a toolkit for reliable messaging. Support both UDP and TCP. Failure detection: crashed nodes are excluded from the membership. Flow control to prevent slow receivers to get overrun by fast senders. Compression.

2 Starting a cluster

main class is JAC-A-BOO server, retrieving information about computational nodes.

To allow multiple users to run JAC-A-BOO at the same time on the same nodes, using different versions, codes and configurations, each run chooses a random port in the [4915265535] range, which is the port interval that is reserved to non-standard applications.

An JAC-A-BOO server does only one thing: it executes each computation requests it receives. Each time returning the result to the client of the request.

2.1 Getting information about computational nodes

JAC-A-BOO comes with only one pre-defined computation request: the one that retrieves information about the node that fulfills it.

Information about cluster nodes is stored in Node objects. Before the cluster is started, the only information available about a node is its name (as wel as its IP address in the case of IP networking). Immediately after JAC-A-BOO server listens for incoming computation requests, the deployer submit its first request: obtaining information of the node itself.

3 Node Information

3.1 always fresh

Node information have a peremption date: after it expires, JAC-A-BOO considers the information out-of-date. As a consequence, the next request (via accessors of the Node class) of node information results in the submission of computation request that will return fresh information.

3.2 Information content

Node information include:

- the number of processors/cores the node has;
- the architecture of these processors/cores;
- the name of the operating system (OS) the node runs;
- the size of Random Access Memory (RAM) the node has;
- the current load average of the node, as defined by the Unix standard;
- the number of requests the node has fulfilled in the past;
- the number of requests the node is currently fulfilling;
- the number of requests the node has in queue;
- some performance benchmarks about the node, including;
  - comparative speed of integer operations;
  - comparative speed of floating-point operations;

– comparative speed of RAM read/writes;
– comparative speed of local Hard Disk Drive (HDD) read/writes.

Only the fourth first items remain unchanged accross subsequent information requests. All others are re-evaluated.

This information enables dynamic load balancing accross the cluster as well as the dynamic selection of the best-matching node for any specific request.

4 Programming model

A *computation request* is a Java object which as the following properties:

- it is serializable, enabling its transport throught network links;
- its class implements the `compute()` method, whose return type is also serializable.

A request can embed data as any non-static/transient field. Such fields will be serialized along with the `ComputationRequest` object, hence by transferred to the node's servers.

Important note: a request cannot be defined as an anonymous class because it must be a static class (which anonymous classes are not). Declaring a request as a non-static class would result in the serialization of its entire enclosing class, which is not desired in most situations.

4.1 The most basic computation request

As a first example, consider the following computation request that returns the user login name of the user running the server. This request carries and processes no data. Instead it returns some information found on the remote node.

```
1 class UserNameRequest extends ComputationRequest<String>() {
2
3         @Override
4         public String compute() {
5                 return System.getProperty("user.name");
6         }
7 }
```

The `ComputationRequest` class is parameterized with the type of the result of the computation.

4.2 Submitted a request

This request can now be submitted to any node, like this:

```
1 UserNameRequest r = new UserNameRequest();
2 String u = r.runOn("grimaudin");
```

4.3 Passing parameters to a request

Such querying requests are a minority. Most computation requests will require the processing of some input data. This input data is embedded in the request object in the form of non-static/transient fields (that the `compute()` method has access to). As such, it is serialized and transferred through the network links along with the request object.

This second example requests the computation of the length of a given string.

```
1 class StringLength extends ComputationRequest<Integer> {
2         String text;
3
4         @Override
5         public Integer compute() {
6                 return text.length();
7         }
8 }
```

This is how this request should be submitted to node *grimaudin*:

```
1 StringLength r = new StringLength();
2 r.text = "Hi there!";
3 int l = r.runOn("grimaudin");
```

4.4 Getting feedback from a remotely running request

A running request can send feedback to its client by calling the method `sendFeedbackToClient(Object feedback)` within the `compute()` method.

Feedback can then be captured by overriding the method `feedbackReceived(Object o)`.

This feature can be used to monitor the progression of a request.

4.5 Dealing with remote failures

If a request fails at running on a node, JAC-A-BOO transfers the Java exception back to the client. This enables the client to handle errors of remote executions as if the they were local, using `try-catch` control structures.

4.6 Synchronous parallel distributed computing

OneNodeOnJob

4.7 Batch processing

Batcher

ListBatcher

5 Communication protocols

5.1 Physical-to-transport layers

5.1.1 Ethernet and TCP

By default JAC-A-BOO TCP/IP as it transport layer. Each request submission opens a new socket to the server. Once the request has returned its result to the client, the connection is closed.

Connection reuse Experiments in JAC-A-BOO have shown that reusing the same socket for all requests submissions improves performance by a factor 1.3. However, when using connection reuse between a client and a server, multiple requests coming from this client are processed sequentially by the server.

5.1.2 Ethernet and UDP

Experiments have shown that using UDP instead of TCP speedups the communications by a factor 2.

5.1.3 Infiniband

JAC-A-BOO supports Infiniband on Linux and Windows nodes. Infiniband is enabled by the deployment of its cluster-specific configuration file on every node, prior to the execution of the virtual machines..

http://docs.oracle.com/javase/tutorial/sdp/sockets/index.html

http://www.infoq.com/articles/Java-7-Sockets-Direct-Protocol?utm_source=infoq&utm_medium=popular_links_homepage

http://pauldone.blogspot.fr/2012/10/writing-your-own-java-application-on.html

Design of scalable Java message-passing communications over InfiniBand http://www.des.udc.es/~juan/papers/JoS2012-2.pdf

5.2 Application layer

5.2.1 Efficient request transfer

The byte-representation of computation request is achieved by means of serialization. JAC-A-BOO supports the serialization algorithm coming with the standard Java library, as well as Google FST. When using the standard Java serialization, a single request with no data embedded is represented by a chunck of 158 bytes. This size is reduced to 15 bytes when using FST, hence a dramatic reduction factor of about 10. When serializing multiple requests in one shot, this factor reduces to 4.

But JAC-A-BOO goes further to enhance network throughput.

5.2.2 Compressed communications

Compressing networked communication introduce the following compromise: compressing reduces the amount of data in transit on the network, thereby fastening communications, but compressing data takes time.

In TCP networking, compressing a message that is smaller than a TCP frame is a waste a time because frames have a fixed size. Any message smaller than a TCP frame imposes the transmission of a complete TCP frame anyway.

Compression is desirable when it the reduction of the transmission duration resulting from the reduction of the number of frames transmitted exceed the time overhead due to the compression process itself.

For example, if a message is slightly bigger than a TCP frame, compressing it will likely permit to transmit it embedding in one single TCP frame, thereby speeding up communication by a factor 2. The benefit of compression depends on the content of the messages. It is then application-specific. Highly compressable messages can result in much higher compression factors, resulting in much higher network throughput.

Experiments on JAC-A-BOO, show that a set of computation requests (with no data embedded in them) is compressed by a factor 60, regardless of the serializer used.

JAC-A-BOO support GZIP compression. Work in progress to support Snappy, the compression algorithm using in JTable and Google RPC mechanisms.

6 monitoring graphical user interface

7 Installation

Download the JAR file and add it to your CLASSPATH.