



Parallel Java Course

From thread to cloud

Paul Guernonprez

Intel Software and Solutions Group

July 6, 2009

Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Goals

Goals of this course :

- ▶ Learn the basic concepts of parallelism
- ▶ Experience the basic low level implementation : Threads
- ▶ Then the functional programming approach : JSR-166y
- ▶ Test the higher levels : cloud frameworks
- ▶ Understand the difference between controller and model based frameworks



Why you need parallel processing

Multicore vs Manycore

We are in the "multicore" era, the future will be "manycore". Physics of thermal dissipation won't change in the near future, so be prepared to deal with a lot of cores.

For your typical desktop too

With 2 cores on your system, a typical word processor could afford not to be parallel and use one of the two cores. With 64 mini cores it won't the case anymore. Parallel computing is not for High Performance Computing anymore, it's for everybody.

But don't worry, there's simple ways to parallelize.



Threads and Processes

Threads and processes are both OS objects.

Processes

When you launch a client software, it's typically 1 process. Want to use multiple cores ? launch multiple instance of your application.

- ▶ Pro : Processes don't have to worry about other processes code and variables. Very simple to use : launch several times your software (potentially on several machines).
- ▶ Con : The communication between processes is possible but complex and slow. A process is a "costly" object for the OS.

That's why it's a good container for independant tasks.

Example : Apache serving different clients from different processes.



Threads and Processes

Threads

When you launch a client software, there's typically one thread in your process. If you want multiple threads you have to code it.

- ▶ Pro : Very fast and easy communication between threads. Rather light object for the OS.
- ▶ Con : Sharing a variable can result in it's corruption if not handled correctly. You have to know how to protect your variables. Communication is only possible inside a single process (and a machine memory space).

It's the good solution when you have to communicate a lot between parallel tasks.

Example : Applying a filter to different parts of the same image.

*Both processes and threads will use multiple cores.
You have to choose the right level for your problem.*



Why it's easier than you think

Stay abstract

It's important to know how it works at the low level, but you will probably not deal with threads directly and be able to stay at a high level of abstraction and modelisation.

Frameworks

There's a lot of frameworks and libraries to help you develop parallel applications. Some are using threads, other threads and processes. It's important to know before you begin using them, to know what to expect in terms of complexity, scalability and corruption risks.



Parallel Model, Computation or Logic ?

Computation parallelization

Early methods were focused on computation.

Like a matrix inversion computed on several processors.

Model parallelization

Some frameworks allow you to store your data on a large number of computers and access it as a unique resource. Like Google big table of keywords. A simple database is typically parallel by nature.

Logic parallelization

In enterprise environments, you may not have a lot of clients, but each client can potentially generate a huge request and expect the result instantly. If this request is more a matter of logic than data crunching, you may have to parallelize the logical treatment.

What do you need to parallelize your software ?



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



In this course

We will cover :

- ▶ Explicit Threading in Java
- ▶ JSR-166y : Higher abstraction. Functionnal Programming.
- ▶ Hadoop : Mainly model parallelization. Very large scale.
- ▶ Google App Engine : Simple and scalable MVC for simple needs.

Sorry for the extreme schematisation ...



Labs

Labs :

- ▶ License : The development environment for the labs is available from sourceforge under L-GPL.
- ▶ URL : <http://ParaJava.sf.net> (SVN Versioning server)
- ▶ Hardware : It's better to run the labs on a multicore machine.
- ▶ Software : All the labs are in Java using Eclipse "Ganymede" or Ant directly. The Java version of Eclipse is enough for all the labs.
- ▶ OS : The slides and labs are OS independant. But a couple of installation notes focus on linux (Ubuntu 9/04 desktop).
- ▶ Very simple labs : We will study how to solve a very simple and unrealistic problem with several technologies. The problem by itself is not interesting but should be simple enough to understand the differences between the technologies used.



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Thread or Runnable

You have the choice between :

- ▶ extending "java.lang.Thread". All you have to do is place your parallel work in a "run()" method and call "start()".
- ▶ implementing "java.lang.Runnable" (if you are already extending another class) and giving the object to a Thread.



Exemple with Thread

```
// extends java.lang.Thread
public class MyThread extends Thread {

    // place your code to parallelize in "run()"
    public void run() {
        System.out.println("Hello from another Thread");
    }

    public static void main(String args[]) {
        System.out.println("Hello from the main Thread");
        // instanciate a MyThread and call "start()" ...
        (new MyThread()).start();
    }
}
```



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Lab 1

Problem

Let's start from "HelloThreadsSerial.java" : we are calculating the sum of all the numbers between 0 and a variable. For 4 it's $4+3+2+1=10$.

Parallelization

It's implemented as a very simple loop calling "add(int a, int b)".

I could parallelize it having one core compute $4+3=7$ and another one $2+1=3$, then getting the local sums from the two cores to compute the total sum.

Implementation

To do so I could create Threads, and have each Thread compute a subpart of the total loop and store the result in a local sum variable.

The solution is "HelloThreadsSolution.java".



Lab 1 - Before

```
int totalResult= 0;
for (int i = 1; i <= data; i++) {
    totalResult = add(totalResult,i);
}
```



Lab 1 - After : Thread extended

```
private static class ComputeTask extends Thread {  
  
    // define part of the array to process  
    private int startIndex;  
    private int endIndex;  
    // and local result  
    private int result;  
  
    // constructor  
    public ComputeTask(int start, int end) {  
        this.startIndex = start;  
        this.endIndex = end;  
    }  
  
    // working part  
    public void run() {  
        ... details in the following slide  
    }  
}
```



Lab 1 - After : run method

```
// working part
public void run() {
    // not the final result, just a local result
    result = 0;
    try {
        // loop with conditions specific to this Thread
        for (int i = startIndex - 1; i < endIndex ; i++) {
            result = add(result, i + 1);
        }
    } catch (InterruptedException e) {
        // there's a new Exception handling specific to parallel work
        // you could stop one Thread from working ...
        printNameLog("Thread stopped too early !");
    }
}
```



Lab 1 - After : launching Threads

```
// launching threads with different input parameters
for (int i = 0; i < numThreads; i++) {
    // new threads, Runnable and a name as arguments
    ComputeTask computeThread = new ComputeTask(
        i * chunk + 1,
        (i + 1) * chunk);
    threads[i] = computeThread;
    // ... and start thread !
    computeThread.start();
}
// after this loop all the threads are working
// using multiple cores
```



Lab 1 - After : result aggregation

```
// waiting for all threads with "join",  
// and reading thread local variable "result"  
int totalResult = 0;  
for (int i = 0; i < numThreads; i++) {  
    // join() means wait for thread completion  
    threads[i].join();  
    totalResult += threads[i].result;  
}
```



Alternatives

A shared variable

In this implementation we had a local variable "result" for each thread, and aggregated the final result at the end.

But we could have used a shared variable for all threads and increment it instead. Protecting the variable get/set with "synchronized" for concurrent access would have been necessary (or better).

Better aggregation

We decided here to wait for all threads to finish in a specific order and get the result after that. With a large number of threads it would not be very efficient.

With Threads, you have to code a lot more to optimize.



Run

Parameters

The serial version will only need one argument : the input number.

The parallel version will only need 2 arguments : the input number and the number of threads (number should to be a multiple of threads).

Change the parameters in build.properties : "threads.data=40" and "threads.num=4" and run "ant threads-parallel" or "ant threads-serial".

Running time

If you have a quad core machine, hopefully the software will take a little more than 4x more time with one thread than 4.

CPU utilization

You should see a CPU utilization of 100% for the serial version process, and 400% on a quad core machine with 4 threads (still one process).

Launch "top" on linux or Task Manager on Windows.



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Threads Conclusion

Code length

Even for a simple example, the parallelization required a notable amount of code. It could have been far worse should you have to set up a queue or balance work between threads.

Code complexity

The code was simple because the approach was simple. Protecting shared variables or work balancing can be very complex to code.

Applications

Threading is low level and can be used for pretty much anything, but you have to code everything. The code is using objects but the way to think your code is not really abstract.

*It's important to know threads, but in Java
it may not be the best way to solve your problem.*



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Threads vs JSR-166y

Standardized Utilities

The main goal behind the standard JSR-166 (introduced as `java.util.concurrent` in Java5) was to help developers think and code high performance parallel software in a standard and simple way.

Functionnal Programming

Going a bit further, the JSR-166y plans to bring a more "functionnal" approach to java parallel programming and very scallable concepts.

Java Versions and VMs

Threads are in the Sun JVM since the beginning, `java.util.concurrent` package is available in Java 5 (improved in 6).

Other JVM are integrating the package too.

Let's use JSR-166y, the future of `java.util.concurrent` !



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Logistics

Code

The code is "HelloParallelArraysSolution" in the package "eu.guermonprez.paul.oss.parajava.parallelarrays".

Same problem and serial code as before : "HelloThreadsSerial".

Libraries

You need the jsr-166 library jars to compile and run (managed by ant). In the near future similar classes should be available in Java7.

Run

To run the software, launch the ant task "parallelarrays-parallel". Only one parameter is needed : the input data ("threads.data").



Functionnal programming - Data Structure

Data Structure = ParallelArray

Instead of trying to split your data in smaller chunks yourself, just keep in your data in a parallel structure called ParallelArray.

"ParallelLongArray" in my case.

You can create this PA from a real array, or create a linked PA to an existing array, or generate it with a "Generator".

```
// array
long[] array = new long[data];
for (int i = 1; i <= data; i++) {
    array[i - 1] = i;
}

// ParallelArray from array
ParallelLongArray parray = ParallelLongArray.createFromCopy(
    array,
    ParallelLongArray.defaultExecutor());
```



Functionnal programming - Computation

Computation = Operator

Define the work you want to do on each element as an Operator.

"Ops.LongReducer" in my case.

An Inline definition is a typical Java way to implement the Interface.

```
// procedure, with inline definition
LongReducer reducer = new LongReducer() {

    public long op(long a, long b) {
        // call to my silly "add" method
        return add(a, b);
    }
}
```



Functionnal programming - Execution

Execution = Executor

All you need to do is to link your PA data and Operator computation with an Executor.

In my case a default "ParallelLongArray.defaultExecutor()" was enough.
And call it with "reduce" !

```
// execute the reduction  
long base = 0;  
long result = parray.reduce(reducer, base);
```



Explanation - Concepts

Divide and conquer

If you have a large number of independent operations to run, a "divide and conquer" algorithm could try to split the work recursively as long as necessary and fill each thread working queue. Then smart algorithms could try to balance the work between threads.

Reduction

A reduction is a specific kind of operation where you need to collect local results to form a global result. Reductions are frequent in parallel programming. The collection from operations can be linear, like a tree or take other forms.



Explanation - What happened

Adaptative

The "reduce" call created a pool of threads adapted to the hardware and splitted the tasks recursively to fill working queues.

Scalable

The splitting concept by itself is capable of using a large number of cores, and the balance between queues is correcting potential imbalances. The result is highly scalable.

Clear, abstract and short

The functional approach of this library allows the developer to write clear, abstract and short code, leaving implementation and runtime details to the library.

Better than threads in our case !



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Performance and flexibility

Performance

The library can benefit from several optimized algorithms to spread computations on several threads and balance work efficiently.

Simplicity

All you have to is use a simple library, no complex framework or application server.



Style, and much more ...

Filters

We've seen a simple example, but the library is very rich. Use of filters for example allows you to select what kind of data you want to link to your computation.

Functionnal style

You need to adopt a new style to use this library. But the functionnal approach should be easy for java devs. Functionnal programming is clearly a major way to develop efficient parallel applications (with JSR-166y or not).



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Concept

Distributed Filesystem and scheduler

Hadoop is based on a very large distributed filesystem. You can also run tasks staying very close to the data to minimize data movements or processing communication.

The scheduler and distributed filesystem are tightly integrated.

Smart fork, smart join too

Hadoop is not only splitting the work, you can also create reduction operations to compute a result from the independent tasks.

Process based and smart queue

Hadoop is process based. Don't expect ultra fast communication between processes, but processes can run on different machines (thousands). You are no longer limited to a single memory space. The scheduler is capable of relaunching failed tasks (very important for large clusters).



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Basics

Goal

First hadoop lab : setup a pseudo cluster and run a simple example.

Logistics

Running as root for testing purposes. Using "hadoop-0.20.0.tar.gz".

Key based no password login

The Main process will connect to remote cluster machine using ssh to launch local processes. To setup a key based login to "localhost" :

```
ssh-keygen -t dsa
cat .ssh/id_dsa.pub >> .ssh/authorized_keys2
ssh-askpass # enter your password ...
# you should be able to "ssh myhostname" /no password.
```



XML Configuration

Insert in conf/core-site.xml

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

Insert in conf/hdfs-site.xml

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

Insert in conf/mapred-site.xml

```
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:9001</value>
  </property>
</configuration>
```



Administration

HADOOP filesystem and services

```
# Create and format a new HADOOP filesystem :
bin/hadoop namenode -format
bin/start-all.sh      # start services
jps                    # listing java processes with names
 19665 Jps
 19500 TaskTracker
 19373 JobTracker # make sure you have this one !
 18881 NameNode
 19225 SecondaryNameNode
 19004 DataNode
```

Web interfaces

- ▶ NameNode <http://localhost:50070>
- ▶ JobTracker <http://localhost:50030>



Input and launching command

From unix FS to HADOOP FS, and replicate

Let's put the content of the folder "conf" as "input" in hadoop. The conf files will be used as regular files, nothing specific.

```
bin/hadoop fs -put conf input
```

Check the log for replication, and browse the HADOOP filesystem :
http://localhost:50070/nn_browsedfscontent.jsp

Launching command

```
bin/hadoop jar hadoop-*-examples.jar \  
  grep input output 'dfs[a-z.]+'
```

It will instruct hadoop to check the methods contained in the jar file "hadoop-*examples.jar", and launch the command : "grep input output 'dfs[a-z.]+'" This command will read data from the input folder and will write results in output.



Output

Job logs

From the OS, you should see several processes running in parallel

```
09/06/26 18:53:40 INFO mapred.FileInputFormat: Total input paths to process : 13
09/06/26 18:53:41 INFO mapred.JobClient: Running job: job_200906261849_0001
09/06/26 18:53:42 INFO mapred.JobClient: map 0% reduce 0%
09/06/26 18:53:52 INFO mapred.JobClient: map 15% reduce 0%
...
09/06/26 18:54:19 INFO mapred.JobClient: map 100% reduce 100%
09/06/26 18:54:21 INFO mapred.JobClient: Job complete: job_200906261849_0001
...
```

From HADOOP FS to unix FS

At the end of the operation you should have a new folder "output" in your HADOOP filesystem.

Let's get the content of this folder and delete it :

```
bin/hadoop fs -get output output
bin/hadoop fs -rmr output
```



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Map/Reduce framework : setup

Let's try to compute our simple sum with HADOOP. One of the ways to write HADOOP applications is to use the Map/Reduce framework. We'll give the input as lines in text files to demonstrate a simple interaction with HDFS. You can find 4 input files in the data folder. The job description is the main method, as HADOOP apps look like regular command line apps :

```
// job definition, reflection to enable remote launch
JobConf job = new JobConf(HelloHadoop.class);
job.setJobName("HelloHadoop");

// logic : define map, combine, reduce classes
job.setMapperClass(Map.class);
job.setCombinerClass(Reduce.class);
job.setReducerClass(Reduce.class);
```



Map/Reduce framework : result classes and IO

The local and final results will travel and be stored as pairs of Key/Value. We have to define precisely their class.

```
job.setOutputKeyClass(Text.class);  
job.setOutputValueClass(LongWritable.class);
```

IO definition : mainly text pipes from files.

The names of I/O folders come from command line arguments.

```
job.setInputFormat(TextInputFormat.class);  
FileInputFormat.setInputPaths(job, new Path(args[0]));  
job.setOutputFormat(TextOutputFormat.class);  
FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

And run !

```
JobClient.runJob(job);
```



Map/Reduce framework : Map

```
// extends the common "base" and implements "Mapper"  
public static class Map extends MapReduceBase implements  
    Mapper<LongWritable, Text, Text, LongWritable> {  
  
    // called for each line of input  
    public void map(LongWritable key, Text value,  
        OutputCollector<Text, LongWritable> output, Reporter reporter)  
        throws IOException {  
        // from a line of text to 2 longs  
        StringTokenizer tokens = new StringTokenizer(value.toString());  
        long a = Long.parseLong(tokens.nextToken());  
        long b = Long.parseLong(tokens.nextToken());  
        // computation : calling our cpu intensive "add"  
        long sum = add(a, b);  
        // collection, key = "sum" value = the sum  
        output.collect(new Text("sum"), new LongWritable(sum));  
    }  
}
```



Map/Reduce framework : Reduce and Combine

```
// extends the common "base" too but implements "Reducer"
public static class Reduce extends MapReduceBase implements
    Reducer<Text, LongWritable, Text, LongWritable> {

    // called for each key/value generated by "collect"
    public void reduce(Text key, Iterator<LongWritable> values,
        OutputCollector<Text, LongWritable> output, Reporter reporter)
        throws IOException {
        int sum = 0;
        // reduce will sum independant results
        // for each line of input locally,
        // and a global result from all the local results
        while (values.hasNext()) {
            sum += values.next().get();
        }
        // as there's only one key "sum",
        // the value will be our final result
        output.collect(key, new LongWritable(sum));
    }
}
```



Run

In eclipse/ant, build the jar from source with the task "build-jar".
The server is running as root, we would like to use different rights for the data files.

```
# Let's create a new user folder for "paul"
./bin/hadoop fs -mkdir /user/paul
# And change rights as root
./bin/hadoop fs -chown paul /user/paul
# keep your root console open, and run the rest as USER paul,
# put the input files folder on HDFS
# (replace $ECLIPSE_HADOOP_PROJECT with your value)
./bin/hadoop fs -put \
  $ECLIPSE_HADOOP_PROJECT/data paralleljavacourse
# and run
./bin/hadoop jar \
  $ECLIPSE_HADOOP_PROJECT/tmp/dist/pjc-hadoop.jar \
  eu.guermonprez.paul.oss.parajava.hadoop.HelloHadoop \
  paralleljavacourse output
```



Logs

```
09/07/03 16:51:18 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
09/07/03 16:51:19 INFO mapred.FileInputFormat: Total input paths to process : 4
# 4 input files, OK !
09/07/03 16:51:19 INFO mapred.JobClient: Running job: job_200907031650_0001
09/07/03 16:51:20 INFO mapred.JobClient: map 0% reduce 0%
09/07/03 16:51:36 INFO mapred.JobClient: map 50% reduce 0%
09/07/03 16:51:45 INFO mapred.JobClient: map 100% reduce 0%
# map first
09/07/03 16:51:48 INFO mapred.JobClient: map 100% reduce 16%
09/07/03 16:51:54 INFO mapred.JobClient: map 100% reduce 100%
# and reduce
09/07/03 16:51:56 INFO mapred.JobClient: Job complete: job_200907031650_0001
09/07/03 16:51:56 INFO mapred.JobClient: Counters: 18
09/07/03 16:51:56 INFO mapred.JobClient: Job Counters
09/07/03 16:51:56 INFO mapred.JobClient:   Launched reduce tasks=1
09/07/03 16:51:56 INFO mapred.JobClient:   Launched map tasks=4
09/07/03 16:51:56 INFO mapred.JobClient:   Data-local map tasks=4
09/07/03 16:51:56 INFO mapred.JobClient: FileSystemCounters
09/07/03 16:51:56 INFO mapred.JobClient:   FILE_BYTES_READ=62
09/07/03 16:51:56 INFO mapred.JobClient:   HDFS_BYTES_READ=107
09/07/03 16:51:56 INFO mapred.JobClient:   FILE_BYTES_WRITTEN=270
09/07/03 16:51:56 INFO mapred.JobClient:   HDFS_BYTES_WRITTEN=8
09/07/03 16:51:56 INFO mapred.JobClient: Map-Reduce Framework
09/07/03 16:51:56 INFO mapred.JobClient:   Reduce input groups=1
09/07/03 16:51:56 INFO mapred.JobClient:   Combine output records=4
09/07/03 16:51:56 INFO mapred.JobClient:   Map input records=20
...
```



Output

We used files as input and output. The result is in a HDFS file "output/part-00000".

```
# To display the output file directly in HDFS
./bin/hadoop fs -cat output/part-00000
sum      820
# OK !
# you can get a copy of the folder from HDFS :
./bin/hadoop fs -get output /tmp/output
# remember to delete the output folder before you relaunch
./bin/hadoop fs -rmr output
```



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Conclusion

Data based

With hadoop you have a highly distributed filesystem where you could store huge amounts of data. The computations are associated with the data where it is to minimize movements.

Data and compute, that's all

HADOOP is just a component in a typical enterprise system. You won't find a friendly authentication system or an associated vue.



Conclusion

Highly scalable ...

You can scale to huge amounts of data or size of job on your cluster. The reduce/combine will let you get a single result from a lot of mapped computations.

... but rather simple by default

You can scale, but you don't have a lot of distribution mechanisms and frameworks by default.

*Scalable, but you'll have to code to solve your specific problem.
It's more a development framework than a simple library.*



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Concept

MVC J2EE

It's a complete Model-View-Controller framework. If you know Java Enterprise you'll quickly understand the concept and packaging. There's other interfaces to the application server (like Python) but we'll work with Java.

Google Big Table vs Entity Beans

Instead of writing and running J2EE entity beans for persistence, you can access Google Big Table from a typical Java persistence system : JDO.

Authentication

Instead of a standard J2EE authentication system, you can use Google logins.

It's simpler than a typical J2EE server, but easier too.



Built to scale

Simple

Big table is a very simple persistence system to store simple information, you won't be able to store easily very complex types or extend the database engine like your typical heavy database system.

Scalable

Big Table is built to scale, a unique table interface is seen from all datacenters all over the world.

Tradeoff

That's why you can't have incremental UIDs, or have an exact count of items in your table.

Built to scale, but you may have to change a few things.



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



First Project

GAE eclipse plugin

If you work with eclipse you may install a GAE plugin from "software updates".

Create a project

Create a new project called "ParallelJavaCourseGAE" in the "Google/Web Application Project" menu. (GAE only, no GWT)

Container : WAR

The project created is a typical J2EE WAR archive, with a servlet in "src" and the basic files in "war" : index.html web.xml appengine-web.xml

Server

A test server is included. You'll later deploy your app on Google servers.



First Run - What Happened

Run

Menu "Run/Run As/Run As Web Application".

Browse `http://localhost:8080`

You'll see an index file, then a "hello world" page.

What happened ? `web.xml`

The `web.xml` is a standard web-app definition form, defining in our case :
servlet, mapping and welcome file.

What happened ? `index.html` and mapping

The default entry point, the page is showing a link.

The link is mapped to the servlet.

What happened ? Servlet

The servlet is the Java code, using the output writer to send "Hello, world" to your browser.



Improvements - Login

Checking authentication

In the servlet, we are using a GAE specific "UserService" to check if the user is authenticated, and if not (user is null) forwarding to the login page (automatically generated).

```
// authentication service and user from service
UserService us = UserServiceFactory.getUserService();
User currentUser = us.getCurrentUser();

// null if not authenticated
if (currentUser != null) {
    resp.setContentType("text/plain");
    resp.getWriter().println("Hello, " + currentUser.getNickname());
} else {
    // creating a redirection to the auth service
    // with the URL as argument to come back
    resp.sendRedirect(us.createLoginURL(req.getRequestURI()));
}
```



Improvements - Persistent objects

Java Data Objects

JDO is a very simple way to store your objects : you create a Class "UserRequest" and write special "@Persistent" annotations before the properties. Everything will be managed at runtime : each instance of your class will be stored with properties.

```
// A special annotation
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class UserRequest {
    // id is a special property, it's the primary key
    @PrimaryKey
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Long id;

    // regular property
    @Persistent
    private User client;

```

...



Improvements - Persistent objects

Java Data Objects

The constructor is very simple, the rest is just couples of getters/setters for properties (just a getter for the primary key).

```
// Constructor with regular arguments
public UserRequest(User client, Date date, Long request_from,
Long request_to) {
    this.client = client;
    this.date = date;
    this.request_from = request_from;
    this.request_to = request_to;
}
// getter for id
public Long getId() { return id; }
// getter and setter for properties
public User getClient() { return client; }
public void setClient(User client) { this.client = client; }

...
```



Improvements - Persistence management

To make an object like `UserRequest` persistent, we need to use a `PersistenceManager` (created from a `PersistenceManagerFactory`). To retrieve objects, we can query this `PersistenceManager` with a SQL like interface.

```
// Persistence Manager from static factory
PersistenceManager pm = pmf.getPersistenceManager();
// Creating an object, making it persistent
UserRequest userReq = new UserRequest(currentUser, new Date(),
    data_from, data_to);
pm.makePersistent(userReq);
// Querying
String query = "select from " + UserRequest.class.getName()
    + " order by date desc ";
List<UserRequest> requests = (List<UserRequest>)
    pm.newQuery(query).execute();
if (!requests.isEmpty()) {
    for (UserRequest request : requests) {
        resp.getWriter().println(" - " + request);
    }
}
```



Improvements - Computing part

The computing part is the same "add(a,b)" method.

But we want to submit different parts of the computation to different servlet instances (different web requests). So we create an html form in the "index.html" page and add a couple of lines in the servlet to get the variables.

```
<form action="paralleljavacoursegae">  
  Choose an interval to compute :<br/>  
  <input name=data_from value="1"/><br/>  
  <input name=data_to value="40"/><br/>  
  <input type=submit value="Compute"/>  
</form>
```

```
// Computing the result  
Long data_from = new Long(req.getParameter("data_from"));  
Long data_to = new Long(req.getParameter("data_to"));  
int totalResult = 0;  
for (int i = data_from.intValue(); i <= data_to.intValue(); i++) {  
  totalResult = add(totalResult, i);  
}
```



Deploy and run locally

Local or Remote server

You can run the application locally, as a simple application server is provided with the plugin. The storage and authentication are not the real services you would find on Google servers, but it's good enough to test.

Deploy locally first

To run locally, click on "Run/Run As/Run As WebApplication". You will see the server log in an eclipse console. The typical message is "The server is running at `http://localhost:8080`". Should you want to change the server code, you can stop the server with the stop red button in the console. (To avoid "socket: Address already in use" errors)

Run

Browse to `http://localhost:8080` and you should see the `index.html` page. You can submit a request and see the cpu usage locally.



Deploy and run on Google servers

Account and administration

You need to create a google apps login, then access the admin interface to create an application definition.

Just a technical note ...

For the local example, I create a silly "add()" method to generate heavy cpu usage. Remove the lines before deploying your application on Google servers. You should have :

```
private int add(int a, int b) { return a + b; }
```

Deploy and run on remote servers

Right click on your project, and choose "Google/Deploy to App Engine". Fill the project settings (id and version), your password, and "Deploy". You can now browse to <http://my-app-id.appspot.com> and submit requests.



Run

The final result for 1..40 is $210+610 = 820$

```
# localhost:8080/paralleljavacoursegae?data_from=1&data_to=20
Hello, paul@guermonprez.eu
Answer : 210
Saving your request 1 to 20
All saved requests :
  - Numbers 1/20 requested by paul@guermonprez.eu at Wed Jun 24 16
Done.

# localhost:8080/paralleljavacoursegae?data_from=21&data_to=40
Hello, paul@guermonprez.eu
Answer : 610
Saving your request 21 to 40
All saved requests :
  - Numbers 21/40 requested by paul@guermonprez.eu at Wed Jun 24 16
  - Numbers 1/40 requested by paul@guermonprez.eu at Wed Jun 24 16
Done.
```



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Conclusion - Scalability

Point of parallelization

In this example each request does not take a lot of time to complete. So we kept the serial version for the compute part. The point of parallelization is at the request level. You can submit various requests and build a composite result from them as shown in the (1..20+21..40) example. But this method is very basic and won't provide a smart data balance between requests. It is not the way it's meant to be used. Each request is computing a single result in serial mode.

Scalability

Here we do not try to compute a single huge request in parallel but a myriad of simple requests, each in serial mode. The software will scale as the number of clients and requests increase. The scalability will concern both the storage engine and the computation part.



Conclusion - Services and MVC

Associated services

If you use this kind of framework, you will typically want to have associated services like authentication or messaging. The value of an application server is often due to the associated services more than the engine itself.

MVC

Another aspect to consider is how well your framework will handle the 3 sides of MVC together ? It's always interesting to create a scalable software for number crunching, but if you plan to offer the service on internet, having a scalable visual interface tightly integrated with your logic is important too (not used in this simple example).



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Software architecture - You may ask yourself :

Where do you insert parallelism ?

Is it a very high level, like launching several applications with different input files ?

Or rather a multiture of very simple and short computations ?

How independant are your jobs ?

Do you need to accept a lot of web request from different users focusing on different data sets or is it a single large request managing a single huge set ?

Do you need associated services ?

Do you need associated services like an authentication system for multiple clients ? Do you need a good MVC integration or is it just about number crunching ?



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Typical Java cluster bricks

In Java, some typical components or a scalable architecture are :

- ▶ Reflection : needed for remote execution and dynamic update of the remote code.
- ▶ Remote Procedure Calls : needed to communicate and synchronize different JVMs in your cluster. In Java going from a low level RMI to a high level SOAP messaging is easy. Your software can be deployed with different technologies : from the secured datacenter to the cloud.
- ▶ Style / Fonctional programming : interesting to define your tasks, with a clear model-controller delimitation.
- ▶ Architecture / MVC.



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Typical C cluster bricks

- ▶ Instead of reflection : A single software binary, ran with different input parameters on different machines. Or messages to differentiate.
- ▶ Instead of RPC : Communication by messages, often low level datatypes only.
- ▶ Instead of fonctionnal programming : Threads or serial-MPI.
- ▶ Instead of MVC : Very deep mix of data and logic.



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Different frameworks, different needs :

Threads

Threads

- ▶ Form : Included in the basic Java VM since day one.
- ▶ Hardware : Single memory.
- ▶ Coding : Heavy and low level.
- ▶ You can do everything with threads as long as you code it yourself.



Different frameworks, different needs :

JSR166-y

JSR-166y

- ▶ Form : Library (but JSR standardized) soon in the JVM.
- ▶ Hardware : Single memory.
- ▶ Coding : Light and high level fonctionnal programming.
- ▶ MVC ? : No data/vue at all. Controller, solving mainly computation problems (flow parallelization is very complex).
- ▶ Scalable ? : Low granularity, very scalable (on a single machine).



Different frameworks, different needs :

Hadoop

Hadoop

- ▶ Form : Complex cluster setup.
- ▶ Hardware : Cluster (potentially very large).
- ▶ Coding : Requires a totally different approach, your application is running in a specific managed environment. Rather simple algorithms by default.
- ▶ MVC ? : No vue at all. Model based (rather simple but very scalable), computations based on data are tightly coupled and scheduled.
- ▶ Scalable ? : Very scalable, capable of solving very large problems, or handling a lot of requests (or both).



Different frameworks, different needs :

Google App Engine

Google App Engine

- ▶ Form : Complex cluster setup.
- ▶ Hardware : Cluster (potentially very large).
- ▶ Coding : Requires a totally different approach, your application is running in a specific managed environment. But it looks a typical (slightly simpler) J2EE environment.
- ▶ MVC ? : Complete MVC framework with associated services, data is managed by a very simple but highly scalable storage (Big Table). No system to interact between tasks.
- ▶ Scalable ? : Very scalable, but typically handling a lot of small independant requests.



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Legal

License

This document is released under "GNU Free Documentation License".

Copyright

Copyright Paul Gueronprez for Intel, Paris June 2009.

Trademarks

The Intel logo is a registered trademark of Intel Corporation.



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Links

- ▶ Java and Eclipse :
 - ▶ Java Tutorial
 - ▶ Eclipse : Getting Started
 - ▶ Apache Ant Manual
- ▶ Threads and concurrency :
 - ▶ Java Tutorial : Concurrency
- ▶ JSR166y :
 - ▶ Official Library Page : Examples
 - ▶ IBM "Stick a fork in it, Part 1"
 - ▶ IBM "Stick a fork in it, Part 2"
- ▶ HADOOP :
 - ▶ Official Wiki : MapReduce
 - ▶ Official Wiki : Tutorial
- ▶ GAE :
 - ▶ Getting Started in Java



Introduction

Goals and Concepts

Logistics

Threads in Java

Basics

Lab 1

Conclusion

JSR-166y

Basics

Lab 2

Conclusion

Hadoop

Basics

Lab 3.1

Lab 3.2

Conclusion

Google App Engine

Basics

Lab 4

Conclusion

Final conclusion

Interesting questions

Typical Java bricks

Typical C bricks

Different needs

Legal

Links

Contact



Contact

Feel free to contact me if you see bugs in this document,
or just to tell me how it works for you :

- ▶ Paul Guermonprez
- ▶ email : `mailto:paul.guermonprez@intel.com`
- ▶ `http://softwareblogs.intel.com/author/paul-guermonprez`

