

Edgar Andrés Moncada

0832294

PJ - A High Level Library for Parallel Programming in Java

The purpose of the present workshop is to ground some concepts related with threads, message passing and clustering.

The author assumes that you have read the first four chapters of the “Building Parallel Programs - SMPs, Clusters, and Java” book. This document will guide you to apply some concepts described in that book.

First, the reader would run some examples related with the threads abstraction provided by the Parallel Java library. Then a set of steps for deploying a cluster which would be able to run applications using the suggested library are presented.

But first... answer the following questions

1. Explain in your own words what Figure 1.2 and 1.3 are trying to exemplify.

La figura 1.2 trata de ejemplificar que al tener k unidades de procesamiento, un cierto problema que tenga n elementos a procesar puede terminarse en n/k , mientras que con 1 sola unidad se demorará n . Esto quiere decir que cierto problema puede demorarse menos tiempo a medida que aumente las unidades de procesamiento.

La figura 1.3 trata de ejemplificar que al tener k unidades de procesamiento, un cierto problema que tenga n elementos que tome un cierto tiempo en una sola unidad de procesamiento, para una entrada de $n*k$ tomará la misma cantidad de tiempo en las k unidades. Esto quiere decir que un cierto problema que se aumente el tamaño de los datos a procesar puede realizarse en el mismo tiempo al aumentar de igual forma las unidades de procesamiento.

2. Describe two applications suitable to be run on a parallel platform (Section 1.2). Choose one application I/O intensive and other one which is computationally intensive.

Encontrar coincidencias en secuencias de proteínas es un problema computacionalmente intensivo. Las proteínas están conformadas por diferentes cantidades de aminoácidos y se han identificado 20 de esos elementos. El problema consiste en determinar cierta secuencia para qué puede servir, y la mejor forma de determinarlo es compararlo con otras proteínas. Las secuencias de proteínas pueden tener entre 2 a 35000 elementos, lo que hace esto un problema de muchos cálculos. Además que existen bases de datos con cerca de 385000 proteínas. Por estas razones una plataforma paralela puede atacar mucho mejor este problema.

El problema de los primos Mersenne consiste de números primos que tienen la forma $2^n - 1$. No todos los números en esta forma son números primos. Para determinar si un número es primo requiere de muchos cálculos, sobre todo si este es grande, el problema aumenta cada vez mas y mas exponencialmente, ya que a pesar de que se determine otro máximo primo Mersenne este problema continua sin ningun fin. Este tipo de problema es de E/S intensiva y que fácilmente se puede adaptar a una plataforma paralela.

3. What characterizes a [Symmetric Multi-Processor](#) or Shared Memory Multiprocessor (SMP) system. Explain what the meaning of the “invalidate” label in Figure 2.4.

La característica es que la arquitectura del computador replica la CPU, cada una con su caché, conjunto de instrucciones, funcionales y registros, pero con Memoria compartida. El problema que se tiene es a nivel de la cache, por eso la etiqueta de inválido, puede presentarse el problema que cuando uno de los procesadores lee un cierto espacio de Memoria, el otro poco después puede modificarlo, causando que el dato leído sea inválido.

4. Look at Figures 2.7 and 2.9 then explain what the most relevant difference between them.

Las dos figuras muestran de qué manera correría un programa paralelo en dicha infraestructura.

- Para la figura 2.7 tenemos una arquitectura con varios procesadores o CPUs que comparten una memoria principal, en el caso de la figura 2.9 se tienen nodos computacionales cada uno con sus procesadores y memorias autónomos.
- La comunicación entre CPUs en la figura 2.7 se maneja localmente de manera sencilla, en cambio en la figura 2.9 es necesario que los CPUs de las máquinas o nodos se comuniquen por medio de mensajes ya que estas se comunican de manera externa.
- Para la figura 2.7 las tareas que ejecutan los núcleos son Hilos mientras que en la figura 2.9 estos son procesos que a su vez podrían contener hilos.

5. What do you understand by a Hybrid Parallel Program, Figure 2.11.

Básicamente es un programa que se ejecuta en una arquitectura que combina las computadoras SMP y los Clusters, donde se tienen Nodos con arquitectura SMP. Cada Nodo ejecuta un proceso y este será dividido en hilos que corren paralelamente en cada núcleo del SMP; pero al haber nodos independientes es necesario usar el paso de mensajes de los clusters para comunicar los procesos que se ejecutan. Así, el programa se replica en cada nodo pero cada nodo tendrá datos distintos que procesará, y cada nodo podrá ejecutar los procesos de manera independiente y de ser necesario la

comunicación con otro nodo, este se comunicara por medio de mensajes a través de la red.

6. What do you understand by the following parallel pattern designs

Los patrones de diseño paralelo permiten diseñar problemas que requieren de grandes cantidades de computación. Los pasos para diseñar un programa paralelo usando dichos patrones es:

- Identificar el patrón que mejor se ajusta al problema.
- Tomar el patrón de diseño sugerido como el punto inicial.
- Implementar el diseño usando la construcción apropiada en un lenguaje de programación paralelo.

a. result parallelism

Es el diseño enfocado para problemas que requieren la producción de múltiples resultados. Es decir que cada procesador puede llevar el cálculo completo sobre una estructura de datos y producir su respectiva salida, cada resultado se produce de manera paralela.

b. agenda parallelism

Es el diseño enfocado para problemas que requieren producir uno o pocos resultados a partir de un gran número de entradas. Se tiene un equipo o conjunto de procesadores y una agenda de tareas que deben de ser desarrolladas para resolver el problema, cada procesador puede procesar cualquiera de las tareas. Al final el resultado de interés no será el que arroje cada procesador sino sólo ciertos resultados o una combinación de ellos.

c. specialist parallelism

Este diseño es una variante del Agenda Parallelism, donde se tiene el grupo de tareas a desarrollar para dar solución al problema y el equipo de procesadores. La diferencia es que cada procesador solo puede desarrollar una de las tareas. Además, el trabajo para este tipo de un Agend Parallelism es desarrollar la misma tarea en una serie de items.

7. What is the difference between clumping and slicing?

En Cumpling se piensa en agrupar los procesadores en uno, así cada procesador computa muchos resultados en lugar de uno solo, y en contraste Slicing se piensa en dividir las entradas

en segmentos en tantos como procesadores se tengan, así cada procesador computa un segmento de resultados que le correspondan.

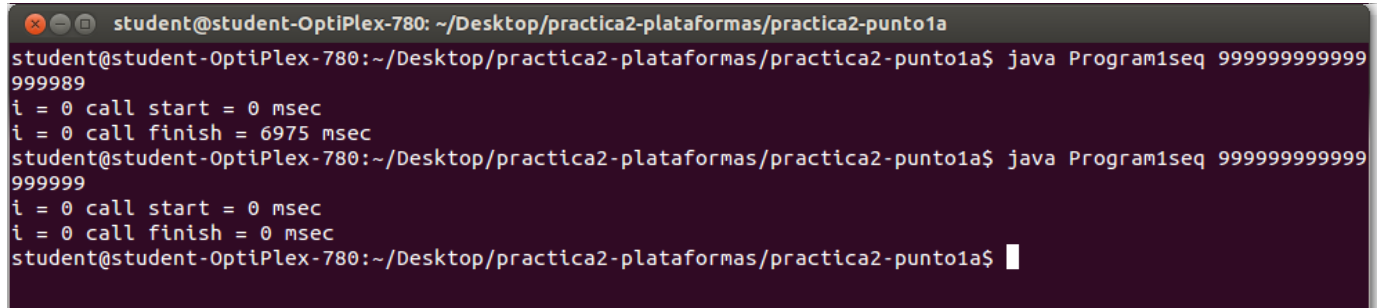
Hands on the code

Las recetas y archivos se encuentran en la carpeta practica2-punto1a

1. Write a program using the code from Section 4.1, run it and jot down your time results.
IMPORTANT: Change the name of the class by **Program1seq.java**.

Se ejecuta el programa smp 2 veces con 1 valor de entrada:

- 999999999999999989: el tiempo de inicio fue en 0 msec y el tiempo de terminación fue de 6974 msec.
- 999999999999999999: el tiempo de inicio fue en 0 msec y el tiempo de terminación fue de 0 msec.



```
student@student-OptiPlex-780: ~/Desktop/practica2-plataformas/practica2-punto1a
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java Program1seq 999999999999999989
9999999
i = 0 call start = 0 msec
i = 0 call finish = 6975 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java Program1seq 999999999999999999
9999999
i = 0 call start = 0 msec
i = 0 call finish = 0 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$
```

Se ejecuta el programa con los valores en el libro:

- 10000000000000037: el tiempo de inicio fue en 0 msec y el tiempo de terminación fue de 220 msec.
- 10000000000000091: el tiempo de inicio fue en 220 msec y el tiempo de terminación fue de 437 msec.
- 10000000000000159: el tiempo de inicio fue en 437 msec y el tiempo de terminación fue de 654 msec.
- 10000000000000187: el tiempo de inicio fue en 654 msec y el tiempo de terminación fue de 871 msec.

```
student@student-OptiPlex-780: ~/Desktop/practica2-plataformas/practica2-punto1a
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java Program1seq 100000000000
0037 1000000000000091 1000000000000159 1000000000000187
i = 0 call start = 0 msec
i = 0 call finish = 220 msec
i = 1 call start = 220 msec
i = 1 call finish = 437 msec
i = 2 call start = 437 msec
i = 2 call finish = 654 msec
i = 3 call start = 654 msec
i = 3 call finish = 871 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$
```

2. Write a program using the code from Section 4.3, run it and jot down your time results.

IMPORTANT: Change the name of the class by **Program1smp.java**.

Note: To compile and run this code download the Parallel Java library from [here](#).

Se ejecuta el programa secuencial 2 veces con 1 valor de entrada:

- 999999999999999989: el tiempo de inicio fue en 5 msec y el tiempo de terminación fue de 6980 msec.
- 999999999999999999: el tiempo de inicio fue en 5 msec y el tiempo de terminación fue de 5 msec.

```
student@student-OptiPlex-780: ~/Desktop/practica2-plataformas/practica2-punto1a
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java -cp ../pj20120620.jar:.
Program1smp 999999999999999989
i = 0 call start = 5 msec
i = 0 call finish = 6980 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java -cp ../pj20120620.jar:.
Program1smp 999999999999999999
i = 0 call start = 5 msec
i = 0 call finish = 5 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$
```

Se ejecuta el programa con los valores en el libro:

- 1000000000000037: el tiempo de inicio fue en 5 msec y el tiempo de terminación fue de 451 msec.
- 1000000000000091: el tiempo de inicio fue en 9 msec y el tiempo de terminación fue de 445 msec.
- 1000000000000159: el tiempo de inicio fue en 11 msec y el tiempo de terminación fue de 443 msec.
- 1000000000000187: el tiempo de inicio fue en 21 msec y el tiempo de terminación fue de 446 msec.

```
student@student-OptiPlex-780: ~/Desktop/practica2-plataformas/practica2-punto1a
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java -cp ../pj20120620.jar:.
Program1smp 10000000000000037 10000000000000091 10000000000000159 10000000000000187
i = 0 call start = 5 msec
i = 0 call finish = 451 msec
i = 1 call start = 9 msec
i = 1 call finish = 445 msec
i = 2 call start = 11 msec
i = 2 call finish = 443 msec
i = 3 call start = 21 msec
i = 3 call finish = 446 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$
```

3. What is the observed speedup?

Para una sola entrada, el programa secuencial y el programa smp se demoran lo mismo, lo cual es lógico dado que es solo un proceso. Para las entradas que dan de ejemplo en el Libro, el programa secuencial toma en resolver todos los parámetros dados en 871; para el programa smp el programa tomo 446. Aunque se puede ver que para cada parámetro se procesa más rápido en el programa secuencial, el tiempo total es más rápido en el programa smp que es lo importante cuando se tienen muchos datos a procesar.

4. Try items 1, 2 and 3 with different number of arguments: 2, 4, 8 and 16. Plot your results.

Para el programa secuencial los resultados son los siguientes:

Para 2 entradas:

```
student@student-OptiPlex-780: ~/Desktop/practica2-plataformas/practica2-punto1a
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java Program1seq 999999999999
999989 999999999999999999
i = 0 call start = 0 msec
i = 0 call finish = 6965 msec
i = 1 call start = 6965 msec
i = 1 call finish = 13928 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ clear
```

Para 4 entradas:

```
student@student-OptiPlex-780: ~/Desktop/practica2-plataformas/practica2-punto1a
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java Program1seq 999999999999
999989 10000000000000187 10000000000000159 999999999999999999
i = 0 call start = 0 msec
i = 0 call finish = 6966 msec
i = 1 call start = 6966 msec
i = 1 call finish = 7182 msec
i = 2 call start = 7183 msec
i = 2 call finish = 7399 msec
i = 3 call start = 7399 msec
i = 3 call finish = 14364 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$
```

Para 8 entradas:

```

student@student-OptiPlex-780: ~/Desktop/practica2-plataformas/practica2-punto1a
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java Program1seq 9999999999
999989 1000000000000187 1000000000000159 9999999999999989 9999999999999989 1000000000000187 100000000000
159 9999999999999989
i = 0 call start = 0 msec
i = 0 call finish = 6957 msec
i = 1 call start = 6957 msec
i = 1 call finish = 7174 msec
i = 2 call start = 7174 msec
i = 2 call finish = 7390 msec
i = 3 call start = 7390 msec
i = 3 call finish = 14348 msec
i = 4 call start = 14348 msec
i = 4 call finish = 21308 msec
i = 5 call start = 21308 msec
i = 5 call finish = 21524 msec
i = 6 call start = 21524 msec
i = 6 call finish = 21741 msec
i = 7 call start = 21741 msec
i = 7 call finish = 28698 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ █

```

Para 16 entradas:

```

student@student-OptiPlex-780: ~/Desktop/practica2-plataformas/practica2-punto1a
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java Program1seq 9999999999
9999989 1000000000000187 1000000000000159 9999999999999989 9999999999999989 1000000000000187 100000000000
00159 9999999999999989 9999999999999989 1000000000000187 1000000000000159 9999999999999989 9999999999
999989 1000000000000187 1000000000000159 9999999999999989
i = 0 call start = 0 msec
i = 0 call finish = 6973 msec
i = 1 call start = 6973 msec
i = 1 call finish = 7190 msec
i = 2 call start = 7190 msec
i = 2 call finish = 7406 msec
i = 3 call start = 7406 msec
i = 3 call finish = 14371 msec
i = 4 call start = 14371 msec
i = 4 call finish = 21338 msec
i = 5 call start = 21338 msec
i = 5 call finish = 21555 msec
i = 6 call start = 21555 msec
i = 6 call finish = 21772 msec
i = 7 call start = 21772 msec
i = 7 call finish = 28739 msec
i = 8 call start = 28739 msec
i = 8 call finish = 35706 msec
i = 9 call start = 35706 msec
i = 9 call finish = 35923 msec
i = 10 call start = 35923 msec
i = 10 call finish = 36140 msec
i = 11 call start = 36140 msec
i = 11 call finish = 43107 msec
i = 12 call start = 43107 msec
i = 12 call finish = 50072 msec
i = 13 call start = 50072 msec
i = 13 call finish = 50289 msec
i = 14 call start = 50289 msec
i = 14 call finish = 50505 msec
i = 15 call start = 50505 msec
i = 15 call finish = 57504 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ █

```

Para el programa secuencial los resultados son los siguientes:

Para 2 entradas:

```
student@student-OptiPlex-780: ~/Desktop/practica2-plataformas/practica2-punto1a
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java -cp ../pj20120620.jar:.
Program1smp 999999999999999989 999999999999999989
i = 0 call start = 5 msec
i = 0 call finish = 7003 msec
i = 1 call start = 5 msec
i = 1 call finish = 7022 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$
```

Para 4 entradas:

```
student@student-OptiPlex-780: ~/Desktop/practica2-plataformas/practica2-punto1a
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java -cp ../pj20120620.jar:.
Program1smp 999999999999999989 1000000000000000187 1000000000000000159 999999999999999989
i = 0 call start = 5 msec
i = 0 call finish = 7163 msec
i = 1 call start = 14 msec
i = 1 call finish = 495 msec
i = 2 call start = 10 msec
i = 2 call finish = 497 msec
i = 3 call start = 30 msec
i = 3 call finish = 7288 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$
```

Para 8 entradas:

```
student@student-OptiPlex-780: ~/Desktop/practica2-plataformas/practica2-punto1a
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java -cp ../pj20120620.jar:.
Program1smp 999999999999999989 1000000000000000187 1000000000000000159 999999999999999989 999999999999999989 1000
000000000000187 1000000000000000159 999999999999999989
i = 0 call start = 6 msec
i = 0 call finish = 15308 msec
i = 1 call start = 6 msec
i = 1 call finish = 392 msec
i = 2 call start = 6 msec
i = 2 call finish = 882 msec
i = 3 call start = 8 msec
i = 3 call finish = 15498 msec
i = 4 call start = 9 msec
i = 4 call finish = 14993 msec
i = 5 call start = 10 msec
i = 5 call finish = 883 msec
i = 6 call start = 14 msec
i = 6 call finish = 881 msec
i = 7 call start = 48 msec
i = 7 call finish = 15401 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$
```

Para 16 entradas:


```
student@student-OptiPlex-780: ~/Desktop/practica2-plataformas/practica2-punto1a
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$ java -cp ../pj20120620.jar:.
Program1smp 99999999999999989 10000000000000187 10000000000000159 9999999999999989 9999999999999989 1000
0000000000187 10000000000000159 9999999999999989 9999999999999989 10000000000000187 10000000000000159 999999
99999999989 9999999999999989 10000000000000187 10000000000000159 9999999999999989
i = 0 call start = 7 msec
i = 0 call finish = 28993 msec
i = 1 call start = 8 msec
i = 1 call finish = 1812 msec
i = 2 call start = 10 msec
i = 2 call finish = 1830 msec
i = 3 call start = 14 msec
i = 3 call finish = 28974 msec
i = 4 call start = 18 msec
i = 4 call finish = 28891 msec
i = 5 call start = 20 msec
i = 5 call finish = 1805 msec
i = 6 call start = 22 msec
i = 6 call finish = 1809 msec
i = 7 call start = 26 msec
i = 7 call finish = 28812 msec
i = 8 call start = 28 msec
i = 8 call finish = 28976 msec
i = 9 call start = 30 msec
i = 9 call finish = 1793 msec
i = 10 call start = 34 msec
i = 10 call finish = 1792 msec
i = 11 call start = 7 msec
i = 11 call finish = 28795 msec
i = 12 call start = 40 msec
i = 12 call finish = 28971 msec
i = 13 call start = 14 msec
i = 13 call finish = 1666 msec
i = 14 call start = 22 msec
i = 14 call finish = 1690 msec
i = 15 call start = 30 msec
i = 15 call finish = 28668 msec
student@student-OptiPlex-780:~/Desktop/practica2-plataformas/practica2-punto1a$
```

En la siguiente tabla se puede observar la comparación de los tiempos entre cada una de las pruebas para cada programa.

| Tamaño de la Entrada | Programa Secuencial (Tiempo Total -tiempo del último parámetro) | Programa SMP (Tiempo Total - tiempo máximo) |
|----------------------|---|--|
| 2 | 13928 | 7022 |
| 4 | 14364 | 7288 |
| 8 | 28698 | 15498 |
| 16 | 57504 | 28974 |

Se puede ver que el programa SMP logra ejecutar los programas en una menor cantidad de tiempo, casi la mitad que la del programa secuencial.

5. Go to the **Program1smp.java** file and take off `n` from line starting with `new ParallelTeam(n).execute`. What happened? Modify the program in such a way

that an arbitrary number of parameters can be processed with the number of cores/processors available in the platform where the program is being run.

Hint: What `Runtime.getRuntime().availableProcessors()` returns?

Las recetas y archivos se encuentran en la carpeta practica2-punto1b

Al quitar el `n`, se crean procesan solamente los dos primeros parámetros, ya que se crean solamente 2 hilos para correr porque solo se tienen 2 núcleos en el equipo. La modificación del programa se adjunta en la carpeta **practica2-punto1b**. El llamado `Runtime.getRuntime().availableProcessors()` devuelve la cantidad de procesadores que pueden ser usados por java.

6. Explain how `ParallelTeam()` and `ParallelRegion()` functions work.

El `ParallelTeam` provee un conjunto o equipo de `K` Hilos, que son especificados en la entrada al construir el objeto, sino se especifica se toma la cantidad de procesadores de la máquina. Este conjunto de hilos, van a ejecutar una `ParallelRegion`, que es una clase abstracta y en la cual se deben definir los métodos `start()`, `run()` y `finish()`, y se pasa al método `execute()` del objeto `ParallelTeam`. De esta manera los hilos ejecutarán las instrucciones en los métodos definidos en su orden.

Clustering

Las recetas y archivos se encuentran en la carpeta clustering.

Now, you shall deploy a cluster. Download the necessary Vagrant + Chef code from [this link](#).

Note: Ask to the instructor for the JDK file.

Note: If you don't have the **precise** box, ask for it too.

Requirements: Vagrant 1.x is required. For upgrading your current Vagrant version run `sudo gem update vagrant`. Source [Base boxes documentation](#).

1. Run the recipes as follows: `vagrant up node1 node2 node3`.
2. When all the machines are up and running, login into master node. From there, run the `ssh` command against `pj-wn1` and `pj-wn2`, e.g. `ssh vagrant@pj-wn1`
You should get a passwordless access. If you have troubles visit this [link](#).
3. To run the cluster, go to this [page](#) and check the "*PJ Cluster Configuration*" section.

```
cd /var/tmp/parajava/  
java -classpath pj.jar edu.rit.pj.cluster.JobScheduler scheduler.conf
```

4. Compile the `Cluster.java` file in run the program as follows
En el master se compila:

javac -cp /var/tmp/parajava/pj.jar:. Cluster.java

- a. java -Dpj.nn=2 -Dpj.np=3 -cp /var/tmp/parajava/pj.jar:. Cluster 67 89 56 72
- b. java -Dpj.nn=2 -Dpj.np=4 -cp /var/tmp/parajava/pj.jar:. Cluster 67 89 56 72
- c. **Jot down your observations**
 - i. What -Dpj.nn and -Dpj.np represent?

a.

```
vagrant@pj-master:~$ java -Dpj.nn=2 -Dpj.np=3 -cp /var/tmp/parajava/pj.jar:. Cluster 67 89 56 72
Job 1, pj-wn1, pj-wn1, pj-wn2
[pj-wn1.cluster.org] 89 is prime
[pj-wn1.cluster.org] 67 is prime
[pj-wn2.cluster.org] 56 is not prime
rank = 1 call start = 24 msec
rank = 0 call start = 42 msec
rank = 0 call finish = 70 msec
rank = 2 call start = 37 msec
rank = 1 call finish = 68 msec
rank = 2 call finish = 60 msec
```

b.

```
vagrant@pj-master:~$ java -Dpj.nn=2 -Dpj.np=4 -cp /var/tmp/parajava/pj.jar:. Cluster 67 89 56 72
Job 2, pj-wn1, pj-wn1, pj-wn2, pj-wn2
[pj-wn2.cluster.org] 56 is not prime
[pj-wn2.cluster.org] 72 is not prime
[pj-wn1.cluster.org] 89 is prime
[pj-wn1.cluster.org] 67 is prime
rank = 3 call start = 62 msec
rank = 2 call start = 29 msec
rank = 3 call finish = 96 msec
rank = 2 call finish = 74 msec
rank = 1 call start = 37 msec
rank = 0 call start = 53 msec
rank = 1 call finish = 102 msec
rank = 0 call finish = 98 msec
```

c.

-Dpj.nn el número de nodos en el back-end y -Dpj.np el número de procesos para ejecutarse el programa paralelo.

<http://www.cs.rit.edu/~ark/pj/doc/edu/rit/pj/PJProperties.html>

Homework

Las recetas y archivos se encuentran en la carpeta **hybrid**.

1. Create a **hybrid** version of the program used for finding prime numbers in such a way that an arbitrary number of prime numbers can be processed in your infrastructure.

Ver el archivo **hybrid/cookbooks/test/files/Cluster.java**

2. Virtual machines used for the **cluster** environment were run in the same physical machine. Take another physical machine and deploy one working node in it and compare against the performance obtained previously. Use different configurations with a different number parameters. Be sure that each parameter be a large number.

Hint: *Some virtual machines should have bridged network interfaces for getting interconnection between physical machines.*

Se crea la siguiente infraestructura usando el mismo Vagrantfile:

1. En una de las máquinas se instala el master y 2 worker nodos.

`vagrant up master`

`vagrant up pj1`

`vagrant up pj2`

2. En otra de las máquinas se instala los otros dos nodos.

`vagrant up pj3`

`vagrant up pj4`

Se configura manualmente el archivo `/etc/hosts` en todos los nodos dado que la ip se asigna mediante DHCP.

```
vagrant@pj-master: ~  
student@student-OptiPlex-780: ~/Desktop/tm... ✕ vagrant@pj-master: ~ ✕  
GNU nano 2.2.6 File: /etc/hosts  
127.0.0.1 localhost  
172.17.9.24 pj-master.cluster.org pj-master  
172.17.9.31 pj-wn1.cluster.org pj-wn1  
172.17.9.21 pj-wn2.cluster.org pj-wn2  
172.17.9.29 pj-wn3.cluster.org pj-wn3  
172.17.9.32 pj-wn4.cluster.org pj-wn4  
[ Wrote 6 lines ]  
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos  
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Luego desde el master se accede mediante ssh para que reconozca los nodos en el momento de enviar los jobs.

```
vagrant@pj-master:~$ ssh vagrant@pj-wn2  
The authenticity of host 'pj-wn2 (172.17.9.21)' can't be established.  
ECDSA key fingerprint is 11:5d:55:29:8a:77:d8:08:b4:00:9b:a3:61:93:fe:e5.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added 'pj-wn2,172.17.9.21' (ECDSA) to the list of known hosts.  
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-23-generic x86_64)  
  
* Documentation: https://help.ubuntu.com/  
Welcome to your Vagrant-built virtual machine.  
Last login: Wed Dec 5 15:07:06 2012 from 10.0.2.2  
vagrant@pj-wn2:~$
```

Luego se puede levantar el cluster:

Luego compilamos y podemos ejecutar el archivo Cluster.java con el resultado:

```
vagrant@pj-master:~$ javac -cp /var/tmp/parajava/pj.jar:. Cluster.java
vagrant@pj-master:~$ java -Dpj.nn=4 -Dpj.np=8 -cp /var/tmp/parajava/pj.jar:. Cluster 6323271 8123219 5231236 74354321 23124 33242 123212 3423425 2123122 3123121 1 3452343 4123125 4323422 1232132 123123223 33453412
```

```
Job 4, pj-wn1, pj-wn1, pj-wn2, pj-wn2, pj-wn3, pj-wn3, pj-wn4, pj-wn4
i = 0 call start = 18 msec, core 0
i = 0 call start = 19 msec, core 0
i = 0 call finish = 18 msec, core 0
i = 0 call start = 52 msec, core 0
i = 0 call start = 51 msec, core 0
i = 0 call start = 42 msec, core 0
i = 0 call start = 39 msec, core 0
i = 0 call finish = 19 msec, core 0
[pj-wn3.cluster.org] 2123122 is not prime
i = 0 call start = 21 msec, core 0
i = 0 call finish = 52 msec, core 0
i = 0 call finish = 51 msec, core 0
i = 0 call start = 57 msec, core 0
i = 0 call finish = 42 msec, core 0
[pj-wn4.cluster.org] 4323422 is not prime
rank = 4 call start = 7 msec
[pj-wn3.cluster.org] 3452343 is not prime
i = 0 call finish = 21 msec, core 0
[pj-wn4.cluster.org] 123123223 is not prime
i = 0 call finish = 57 msec, core 0
rank = 4 call finish = 197 msec
rank = 6 call start = 12 msec
[pj-wn1.cluster.org] 6323271 is not prime
rank = 5 call start = 26 msec
[pj-wn2.cluster.org] 23124 is not prime
i = 0 call finish = 39 msec, core 0
rank = 7 call start = 17 msec
[pj-wn1.cluster.org] 5231236 is not prime
[pj-wn3.cluster.org] 31231211 is not prime
rank = 0 call start = 30 msec
rank = 5 call finish = 200 msec
rank = 2 call start = 17 msec
rank = 6 call finish = 160 msec
rank = 7 call finish = 189 msec
[pj-wn4.cluster.org] 1232132 is not prime
[pj-wn3.cluster.org] 4123125 is not prime
rank = 1 call start = 42 msec
```

Continúa el resultado:

```
rank = 1 call start = 42 msec  
[pj-wn2.cluster.org] 123212 is not prime  
rank = 2 call finish = 187 msec  
[pj-wn4.cluster.org] 33453412 is not prime  
rank = 5 call start = 26 msec  
rank = 3 call start = 34 msec  
rank = 6 call start = 12 msec  
rank = 1 call finish = 178 msec  
rank = 4 call start = 7 msec  
[pj-wn2.cluster.org] 33242 is not prime  
rank = 0 call finish = 133 msec  
rank = 3 call finish = 203 msec  
rank = 5 call finish = 237 msec  
rank = 7 call start = 17 msec  
rank = 2 call start = 17 msec  
[pj-wn1.cluster.org] 74354321 is not prime  
rank = 6 call finish = 195 msec  
rank = 4 call finish = 238 msec  
[pj-wn1.cluster.org] 8123219 is not prime  
[pj-wn2.cluster.org] 3423425 is not prime  
rank = 1 call start = 42 msec  
rank = 2 call finish = 221 msec  
rank = 7 call finish = 225 msec  
rank = 0 call start = 30 msec  
rank = 3 call start = 34 msec  
rank = 1 call finish = 208 msec  
rank = 0 call finish = 181 msec  
rank = 3 call finish = 227 msec  
vagrant@pj-master:~$
```