

Universidad de los Andes

Facultad de Ingeniería
Departamento de Ingeniería de Sistemas
Infraestructura Computacional
Periodo 2022-20



Caso 1 Documentación

Juan José Ramírez Cala - 202013228
Juan Diego Yepes Parra - 202022391

25 de agosto de 2022
Bogotá D.C.

Tabla de contenidos

1	Problema	2
1.1	Enunciado	2
1.2	Especificación	3
2	Solución	3
2.1	Modelo de dominio	3
2.2	Funcionamiento del programa	3
2.2.1	Ejemplo de un flujo de mensaje	5
3	Capturas de ejecución	7
4	Instalación y repositorio	8

1 Problema

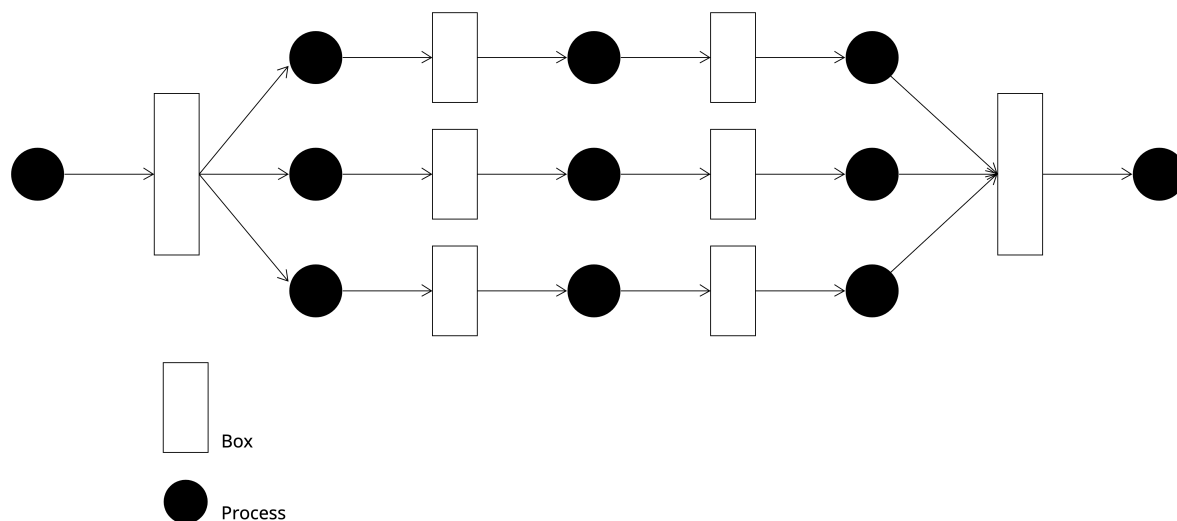
A continuación se detalla el problema a resolver en este caso.

1.1 Enunciado

MapReduce es un modelo de programación ampliamente utilizado para la ejecución de programas paralelos que utilizan grandes cantidades de datos. En este modelo, el conjunto de datos se divide en varios subconjuntos para ser tratados y transformados en paralelo hasta que se vuelven a integrar para producir el resultado final.

El objetivo del caso es diseñar un mecanismo de comunicación para implementar la arquitectura MapReduce. Para este caso, los procesos serán threads en la misma máquina (en realidad debería ser un sistema distribuido; este es solo un prototipo).

Para este ejercicio en particular, se quiere que el modelo funcione como se ilustra a continuación:



Hay tres niveles¹ de procesos, y cada nivel contiene tres procesos que a su vez tienen dos buzones intermedios. El proceso inicial envía N mensajes al buzón inicial, y todos los demás procesos lo modificarán según les llegue el mensaje, para producir outputs del estilo: $M_1T_00T_01T_02$. Este mensaje quiere decir que el mensaje 1 fue transformado en el nivel 0 por los procesos 0, 1 y 2.

¹En este documento se utilizará la palabra nivel diferente del enunciado planteado, ya que allí se hace referencia a los niveles de transformación como la "columna" de procesos. En contraste, nosotros que lo planteamos como la "fila" de procesos, para facilitar la implementación.

1.2 Especificación

A continuación se presenta una especificación (informal) del programa a desarrollar:

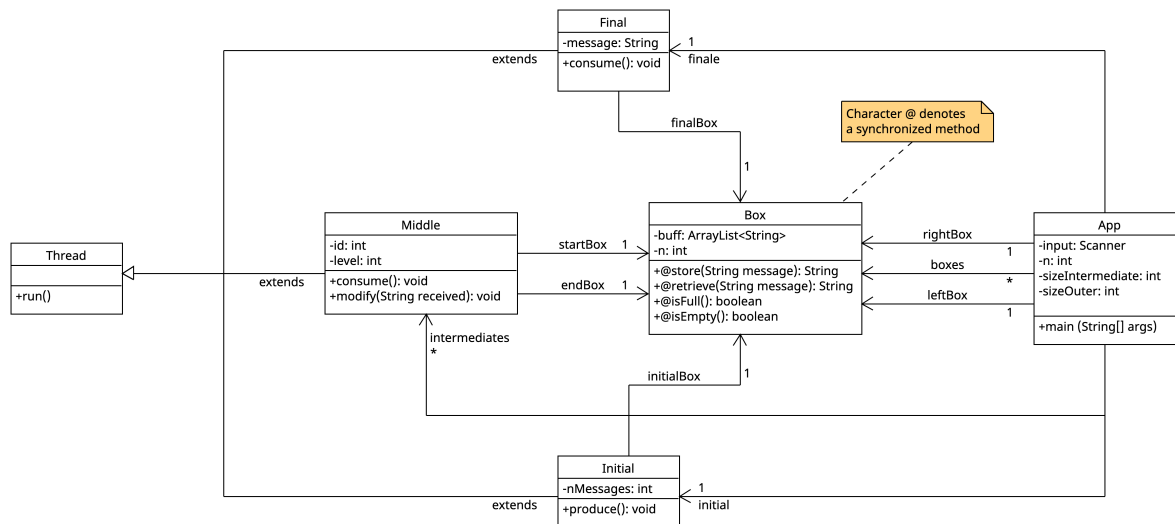
I/O	Nombre	Tipo	Descripción
I	<i>N</i>	int	Número de subconjuntos a generar o de mensajes a enviar.
I	<i>i</i>	int	Tamaño de los buzones intermedios.
I	<i>e</i>	int	Tamaño de los buzones extremos.
O	<i>s</i>	String	Mensaje generado por el flujo de los datos dentro del sistema.

2 Solución

A partir de lo anterior, proponemos la siguiente solución al problema.

2.1 Modelo de dominio

Para el desarrollo del caso se plantea el siguiente diagrama de dominio:



El flujo entre clases y cómo se intercambian información está detallado en la sección 2.2. Por otro lado, se utilizó el carácter @ para denominar que un método es synchronized. Este diagrama y el código están escritos en inglés.

2.2 Funcionamiento del programa

En primer lugar, el programa inicializa en diferentes estructuras de datos la jerarquía y las instancias de cada objeto necesarios para resolver el problema. Con base en lo que el

usuario haya ingresado como parámetros, se crean los buzones Box. (De ahora en adelante se utilizarán los nombres en inglés para cada clase y objeto)

La creación de las estructuras se puede ver aquí:

```

1  Box leftBox = new Box(sizeOuter);
2  Box rightBox = new Box(sizeOuter);
3  for (int i = 0; i < 6; i++) {
4      boxes.put(i, new Box(sizeIntermediate));
5  }

```

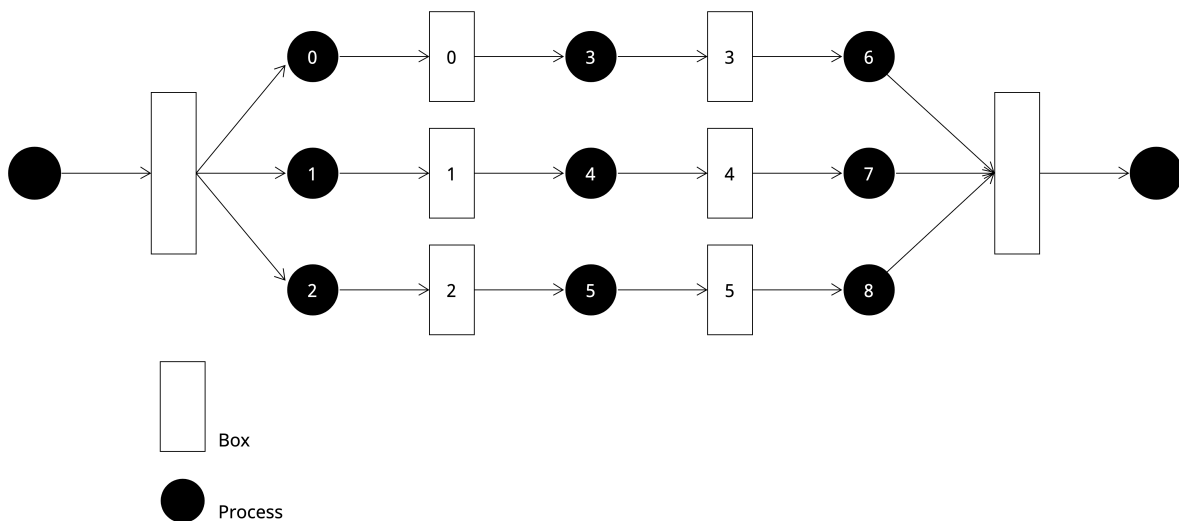
Estos buzones son los que tienen los métodos `synchronized` por lo que son los que contienen los datos compartidos. Las Box de los extremos son creadas en las primeras dos líneas, luego hay un ciclo que crea todas las otras 6 intermedias y las guarda en un HashTable llamado `boxes`. Luego, para instanciar los threads se utilizan las siguientes instrucciones:

```

1  Initial initial = new Initial(leftBox, n);
2  Final finale = new Final(rightBox);
3  for (int i = 0; i < 9; i++) {
4      if (i/3 == 0) { // First row
5          intermediates.put(i, new Middle(i, i%3, leftBox, boxes.get(i)));
6      } else if (i/3 == 1) { // Second row
7          intermediates.put(i, new Middle(i, i%3, boxes.get(i-3), boxes.get(i)));
8      } else if (i/3 == 2) { // Third row
9          intermediates.put(i, new Middle(i, i%3, boxes.get(i-3), rightBox));
10     }
11 }

```

Lo que daría origen al siguiente flujo:



Se crean los procesos `Initial` y `Final`, a los cuales se les asigna su respectivo `Box`. Estos procesos solo necesitan de un buzón, ya que el inicial actúa como un productor y envía

y el final como un consumidor que los recopila. Luego, se crean los procesos intermedios a los cuales se les asigna un Box inicial y final, dependiendo de su columna. Estos procesos Middle actúan como productor y consumidor a la vez, luego tiene sentido asignarles un buzón de entrada y uno de salida.

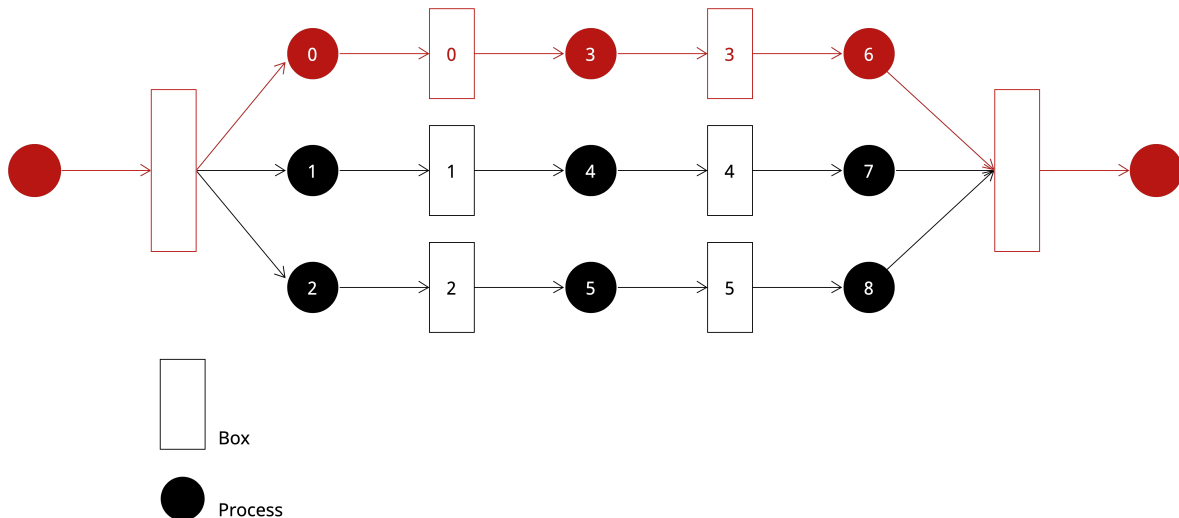
En segundo lugar, el programa empieza a correr todos los threads haciendo `start()` para cada uno así:

```
1  initial.start();
2  for (int i = 0; i < 9; i++) {
3      intermediates.get(i).start();
4  }
5  finale.start();
```

Lo que empezaría a correr todos los threads y a enviar mensajes a los buzones.

NOTA: El enunciado establece un orden para los mensajes un poco diferente al implementado, así que es importante revisar un flujo de mensaje, como se hará en la sección 2.2.1 para entender por qué los mensajes tienen las modificaciones que tienen y cómo se debería ver un mensaje que tuvo determinado flujo.

2.2.1 Ejemplo de un flujo de mensaje



Para mostrar mejor el funcionamiento del programa, supongamos un mensaje `M1`. Este mensaje llegó primero al buzón izquierdo `leftBox` después de haber sido generado por el proceso `initial`, en el método `produce()`; el cual llama al método `synchronized` de su buzón `store()`.

```

1  public synchronized void store(String message){
2      while(buff.size() == n){
3          try {
4              this.wait();
5          } catch (InterruptedException e) {
6              throw new RuntimeException(e);
7          }
8      }
9      buff.add(message);
10     this.notify();
11 }

```

En este caso el parámetro message sería M₁. El método revisa si el Box está lleno en el ciclo. De así serlo, pone a esperar al thread, y si no, agrega el mensaje y notifica a algún thread que puede insertar un mensaje. En este orden de ideas, nuestro mensaje M₁ ya está guardado.

Posteriormente, el thread 0 hace retrieve() de algún mensaje en su Box de entrada, así que supongamos que logra obtener el mensaje M₁. El método funciona así:

```

1  public synchronized String retrieve(){
2      while (buff.size() == 0){
3          try {
4              this.wait();
5          } catch (InterruptedException e) {
6              throw new RuntimeException(e);
7          }
8      }
9      String message = buff.remove(0);
10     this.notify();
11     return message;
12 }

```

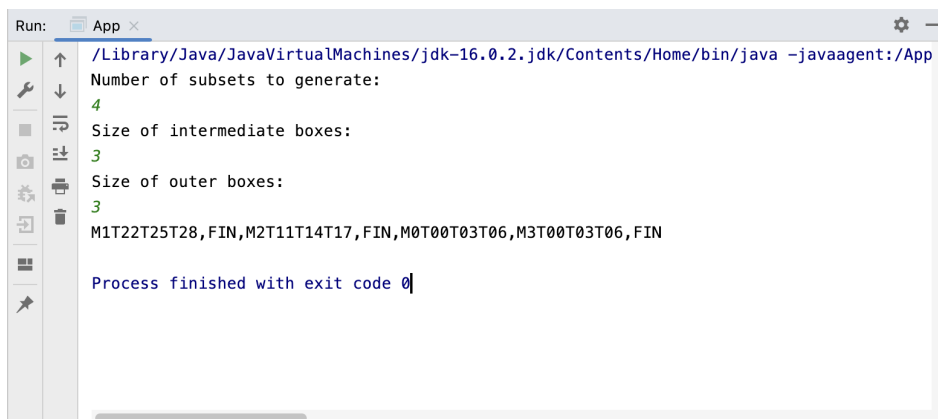
Lo que pasa aquí es que el método pregunta si el Box tiene mensajes, si no tuviera, pone a esperar al thread, pero como si los tiene podemos hacer un remove(0) y retornar el mensaje, a su vez notificándole a otro thread que esté esperando. En este orden de ideas nuestro mensaje ha sido extraído del Box.

Esto funciona iterativamente hasta llegar al thread 6, cuando se envía M₁ (que para este instante ya debe tener todas las transformaciones, luego se debe ver algo como M_{0T00T03T06}), este llega al Box final mediante los métodos ya mencionados y el thread Final concatena todos los mensajes.

Para determinar que ya se han enviado todos los mensajes, el programa envía 3 mensajes de FIN que indican que ya puede terminar con la ejecución de cada thread (y por ende de cada nivel).

3 Capturas de ejecución

A continuación se muestran capturas del programa ejecutándose para demostrar su correcto funcionamiento:



```
Run: App x
/Library/Java/JavaVirtualMachines/jdk-16.0.2.jdk/Contents/Home/bin/java -javaagent:/App
Number of subsets to generate:
4
Size of intermediate boxes:
3
Size of outer boxes:
3
M1T22T25T28,FIN,M2T11T14T17,FIN,M0T00T03T06,M3T00T03T06,FIN

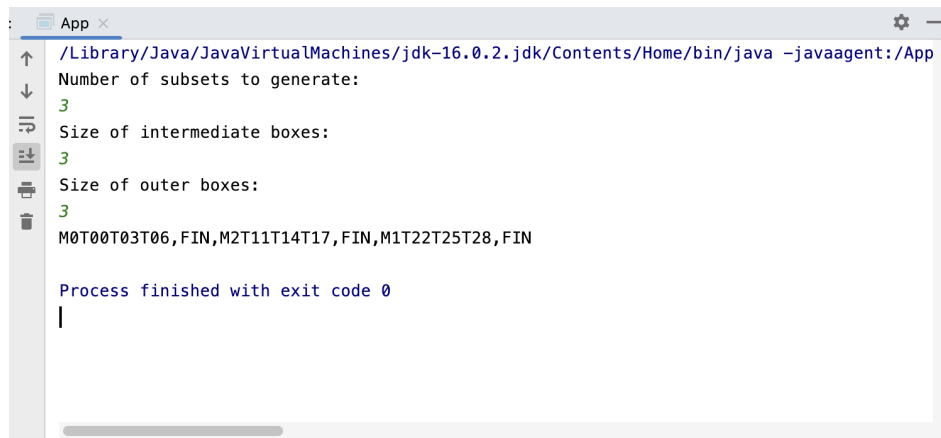
Process finished with exit code 0
```



```
Run: App x
/Library/Java/JavaVirtualMachines/jdk-16.0.2.jdk/Contents/Home/bin/java -javaagent:/App
Number of subsets to generate:
50
Size of intermediate boxes:
10
Size of outer boxes:
5
M2T11T14T17,M1T22T25T28,M4T11T14T17,M0T00T03T06,M3T22T25T28,M5T00T03T06,M8T11T14T17,M6T

Process finished with exit code 0
|
```

1 M2T11T14T17,M1T22T25T28,M4T11T14T17,M0T00T03T06,M3T22T25T28,M5T00T03T06,M8T11T14T17,M6T22T25T28,
M10T00T03T06,M12T11T14T17,M7T22T25T28,M13T00T03T06,M18T00T03T06,M9T22T25T28,M14T11T14T17,
M21T00T03T06,M16T11T14T17,M11T22T25T28,M22T00T03T06,M15T22T25T28,M20T11T14T17,M17T22T25T28,
M27T00T03T06,M33T00T03T06,M23T11T14T17,M19T22T25T28,M38T00T03T06,M24T22T25T28,M41T00T03T06,
M25T11T14T17,M28T22T25T28,M42T00T03T06,M26T11T14T17,M29T22T25T28,M31T22T25T28,M30T11T14T17,
M43T00T03T06,M47T00T03T06,M34T11T14T17,M32T22T25T28,M49T00T03T06,FIN,M37T11T14T17,
M39T11T14T17,M35T22T25T28,M44T11T14T17,M36T22T25T28,M48T11T14T17,FIN,M40T22T25T28,
M45T22T25T28,M46T22T25T28,FIN



```
App x
/Library/Java/JavaVirtualMachines/jdk-16.0.2.jdk/Contents/Home/bin/java -javaagent:/App
Number of subsets to generate:
3
Size of intermediate boxes:
3
Size of outer boxes:
3
M0T00T03T06,FIN,M2T11T14T17,FIN,M1T22T25T28,FIN

Process finished with exit code 0
|
```

4 Instalación y repositorio

Para acceder al repositorio del código fuente diríjase al siguiente [link](#).

Para correr el programa, desde IntelliJ abrir el archivo `App.java` que es el que contiene el método `main()`. Ejecutarlo dando click en Run.