Librería Distribuida para Arrays

Procesamiento Concurrente y Distribuido

CC4P1 Programación Concurrente y Distribuida

André Pacheco

Julio 2025

FACULTAD DE CIENCIAS

Agenda

o

- Objetivo del Proyecto

Agenda

- Objetivo del Proyecto
- Arquitectura y Diseño

Agenda

o

- Objetivo del Proyecto
- Arquitectura y Diseño
- Implementación

Agenda

o

- Objetivo del Proyecto
- Arquitectura y Diseño
- Implementación
- Protocolo de Comunicación

Agenda

o

- Objetivo del Proyecto
- Arquitectura y Diseño
- Implementación
- Protocolo de Comunicación
- Ejemplos de Operaciones

Agenda

o

- Objetivo del Proyecto
- Arquitectura y Diseño
- Implementación
- Protocolo de Comunicación
- Ejemplos de Operaciones
- Tolerancia a Fallos

Agenda

o

- Objetivo del Proyecto
- Arquitectura y Diseño
- Implementación
- Protocolo de Comunicación
- Ejemplos de Operaciones
- Tolerancia a Fallos
- Demostración

Agenda

o

- Objetivo del Proyecto
- Arquitectura y Diseño
- Implementación
- Protocolo de Comunicación
- Ejemplos de Operaciones
- Tolerancia a Fallos
- Demostración
- Conclusiones

Objetivo del Proyecto

o

Desarrollar una librería distribuida

- Arrays distribuidos: `DArrayInt` y `DArrayDouble`

Objetivo del Proyecto

o

Desarrollar una librería distribuida

- Arrays distribuidos: `DArrayInt` y `DArrayDouble`
- Procesamiento concurrente y paralelo

Objetivo del Proyecto

o

Desarrollar una librería distribuida

- Arrays distribuidos: `DArrayInt` y `DArrayDouble`
- Procesamiento concurrente y paralelo
- Comunicación por sockets TCP nativos

Objetivo del Proyecto

o

Desarrollar una librería distribuida

- Arrays distribuidos: `DArrayInt` y `DArrayDouble`
- Procesamiento concurrente y paralelo
- Comunicación por sockets TCP nativos
- Sin frameworks externos

Objetivo del Proyecto

o

Desarrollar una librería distribuida

- Arrays distribuidos: `DArrayInt` y `DArrayDouble`
- Procesamiento concurrente y paralelo
- Comunicación por sockets TCP nativos
- Sin frameworks externos
- Tolerancia a fallos básica

Objetivo del Proyecto

o

Desarrollar una librería distribuida

- Arrays distribuidos: `DArrayInt` y `DArrayDouble`
- Procesamiento concurrente y paralelo
- Comunicación por sockets TCP nativos
- Sin frameworks externos
- Tolerancia a fallos básica

Implementaciones

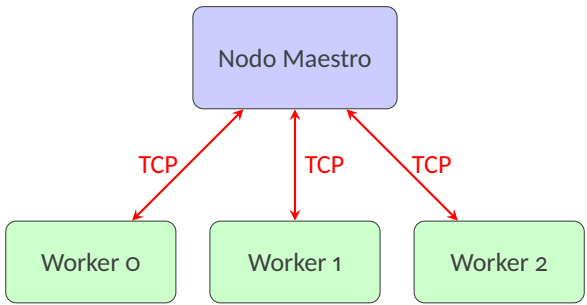
- Java 8+
- Python 3.6+

Arquitectura del Sistema

- o

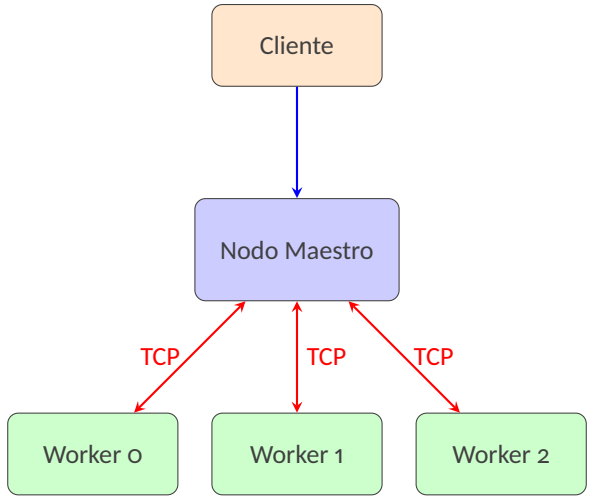
Arquitectura del Sistema

o



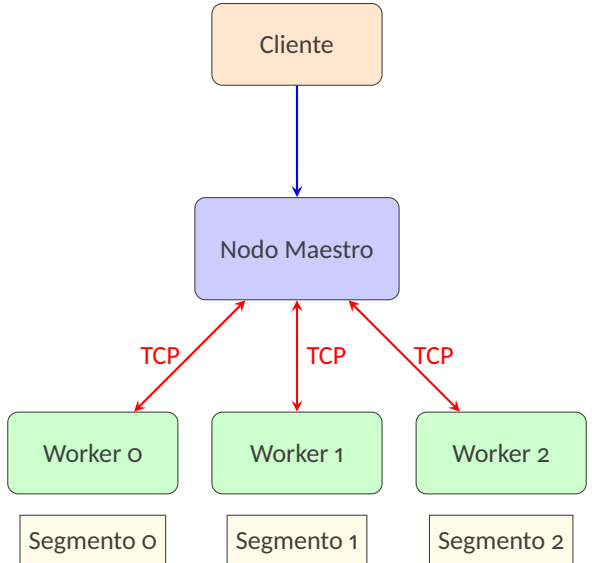
Arquitectura del Sistema

o



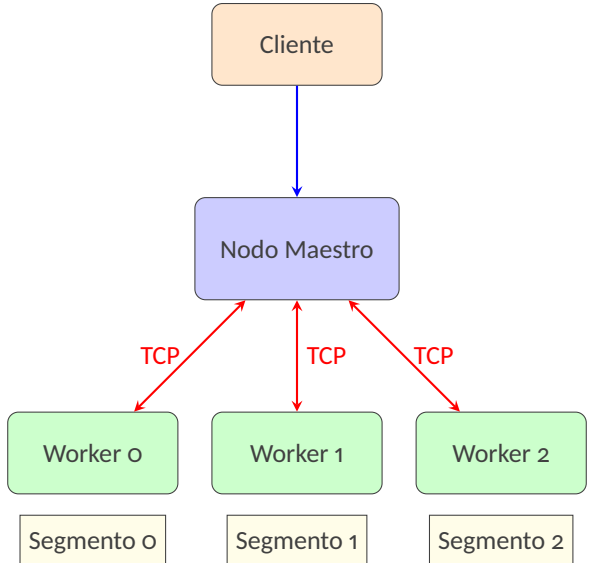
Arquitectura del Sistema

o



Arquitectura del Sistema

o



Implementación - Estructura

o

Java

- `MasterNode.java`

Python

- `master_node.py`

Implementación - Estructura

o

Java

- `MasterNode.java`
- `WorkerNode.java`

Python

- `master_node.py`
- `worker_node.py`

Implementación - Estructura

o

Java

- `MasterNode.java`
- `WorkerNode.java`
- `DArrayInt.java`
- `DArrayDouble.java`

Python

- `master_node.py`
- `worker_node.py`
- `darray.py`

Implementación - Estructura

o

Java

- `MasterNode.java`
- `WorkerNode.java`
- `DArrayInt.java`
- `DArrayDouble.java`
- `Message.java`

Python

- `master_node.py`
- `worker_node.py`
- `darray.py`
- `message.py`

Implementación - Estructura

o

Java

- `MasterNode.java`
- `WorkerNode.java`
- `DArrayInt.java`
- `DArrayDouble.java`
- `Message.java`
- `DistributedArrayClient.java`

Python

- `master_node.py`
- `worker_node.py`
- `darray.py`
- `message.py`
- `distributed_array_client.py`

Segmentación de Arrays

- o

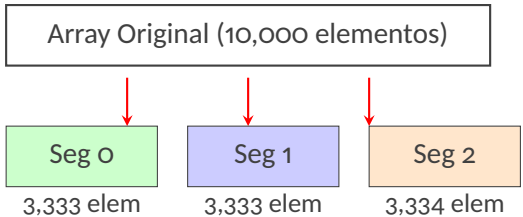
Array Original (10,000 elementos)

Segmentación de Arrays

- Array Original (10,000 elementos)

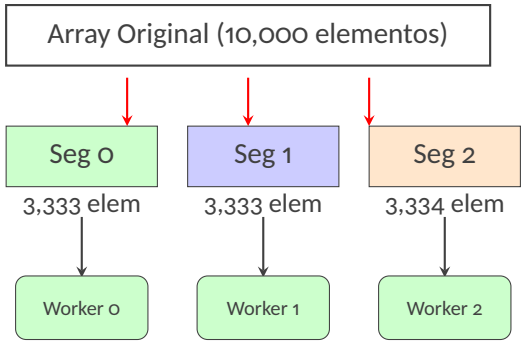
Segmentación de Arrays

o



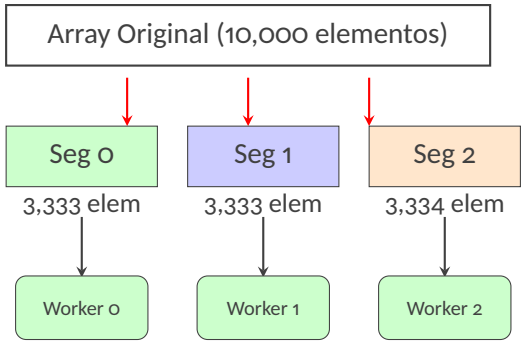
Segmentación de Arrays

o



Segmentación de Arrays

o



Algoritmo de Segmentación

- División equitativa: tamaño_segmento $=$ total$/$workers
- Manejo de residuo distribuido
- Asignación round-robin

Protocolo de Comunicación

o

Formato JSON

```
{
  "type": "MESSAGE_TYPE",
  "from": "NODE_ID",
  "to": "NODE_ID",
  "timestamp": 1234567890,
  "data": {}
}
```

Protocolo de Comunicación

o

Formato JSON

```
{
  "type": "MESSAGE_TYPE",
  "from": "NODE_ID",
  "to": "NODE_ID",
  "timestamp": 1234567890,
  "data": {}
}
```

Tipos de Mensajes

- `REGISTER_WORKER` - Registro de trabajador

Protocolo de Comunicación

o

Formato JSON

```
{
  "type": "MESSAGE_TYPE",
  "from": "NODE_ID",
  "to": "NODE_ID",
  "timestamp": 1234567890,
  "data": {}
}
```

Tipos de Mensajes

- `REGISTER_WORKER` - Registro de trabajador
- `DISTRIBUTE_ARRAY` - Distribución de segmentos

Protocolo de Comunicación

o

Formato JSON

```
{
  "type": "MESSAGE_TYPE",
  "from": "NODE_ID",
  "to": "NODE_ID",
  "timestamp": 1234567890,
  "data": {}
}
```

Tipos de Mensajes

- `REGISTER_WORKER` - Registro de trabajador
- `DISTRIBUTE_ARRAY` - Distribución de segmentos
- `PROCESS_SEGMENT` - Orden de procesamiento

Protocolo de Comunicación

o

Formato JSON

```
{
  "type": "MESSAGE_TYPE",
  "from": "NODE_ID",
  "to": "NODE_ID",
  "timestamp": 1234567890,
  "data": {}
}
```

Tipos de Mensajes

- `REGISTER_WORKER` - Registro de trabajador
- `DISTRIBUTE_ARRAY` - Distribución de segmentos
- `PROCESS_SEGMENT` - Orden de procesamiento
- `HEARTBEAT` - Verificación de salud

Protocolo de Comunicación

o

Formato JSON

```
{
  "type": "MESSAGE_TYPE",
  "from": "NODE_ID",
  "to": "NODE_ID",
  "timestamp": 1234567890,
  "data": {}
}
```

Tipos de Mensajes

- `REGISTER_WORKER` - Registro de trabajador
- `DISTRIBUTE_ARRAY` - Distribución de segmentos
- `PROCESS_SEGMENT` - Orden de procesamiento
- `HEARTBEAT` - Verificación de salud

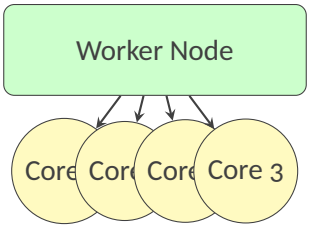
Procesamiento Paralelo

- o

Worker Node

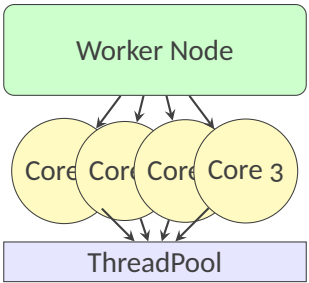
Procesamiento Paralelo

- o



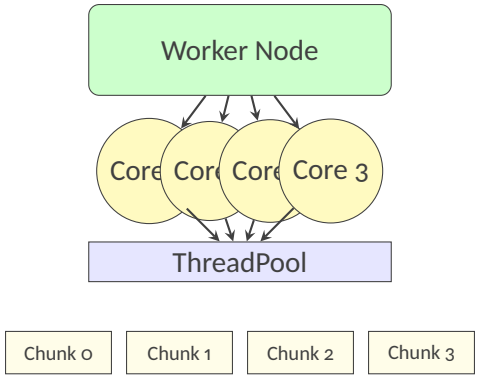
Procesamiento Paralelo

- o



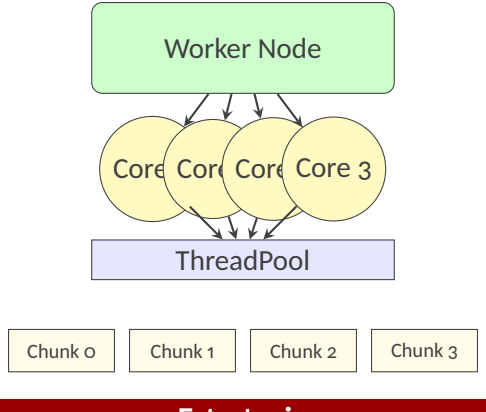
Procesamiento Paralelo

o



Procesamiento Paralelo

o



Estrategia

- Detección automática de núcleos: `Runtime.availableProcessors()`
- División del segmento en chunks
- Procesamiento concurrente con ThreadPool

Ejemplo 1: Operaciones Matemáticas

o

Fórmula

$$\text{resultado} = \frac{(\sin(x) + \cos(x))^2}{\sqrt{|x|} + 1}$$

Ejemplo 1: Operaciones Matemáticas

o

Fórmula

$$\text{resultado} = \frac{(\sin(x) + \cos(x))^2}{\sqrt{|x|} + 1}$$

Implementación Java

- Procesamiento paralelo con ThreadPool
- División del segmento en chunks
- Cada thread procesa su chunk independientemente

Ejemplo 1: Operaciones Matemáticas

o

Fórmula

$$\text{resultado} = \frac{(\sin(x) + \cos(x))^2}{\sqrt{|x|} + 1}$$

Implementación Java

- Procesamiento paralelo con ThreadPool
- División del segmento en chunks
- Cada thread procesa su chunk independientemente

Implementación Python

- Uso de ThreadPoolExecutor
- NumPy para operaciones vectorizadas
- Procesamiento concurrente por chunks

Ejemplo 2: Evaluación Condicional

o

Condición

Si $x \bmod 3 = 0$ o $500 \leq x \leq 1000$:

$$\text{resultado} = (x \cdot \log(x)) \bmod 7$$

Ejemplo 2: Evaluación Condicional

o

Condición

Si $x \bmod 3 = 0$ o $500 \leq x \leq 1000$:

$$\text{resultado} = (x \cdot \log(x)) \bmod 7$$

Procesamiento

- Evaluación condicional para cada elemento
- Aplicación de transformación logarítmica
- Preservación de valores que no cumplen condición

Ejemplo 2: Evaluación Condicional

o

Condición

Si $x \bmod 3 = 0$ o $500 \leq x \leq 1000$:

$$\text{resultado} = (x \cdot \log(x)) \bmod 7$$

Procesamiento

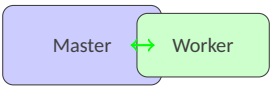
- Evaluación condicional para cada elemento
- Aplicación de transformación logarítmica
- Preservación de valores que no cumplen condición

Resiliencia

- Manejo de excepciones por thread
- Continuación ante fallos parciales
- Consolidación de resultados válidos

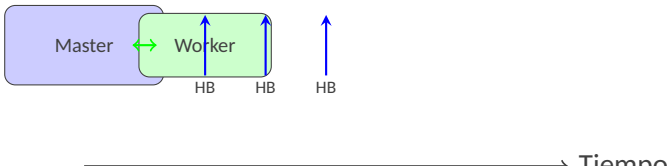
Tolerancia a Fallos

o



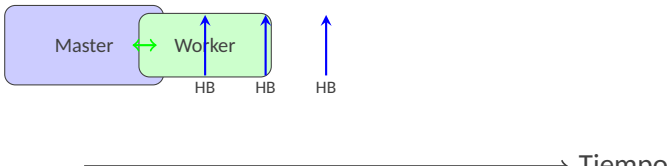
Tolerancia a Fallos

o



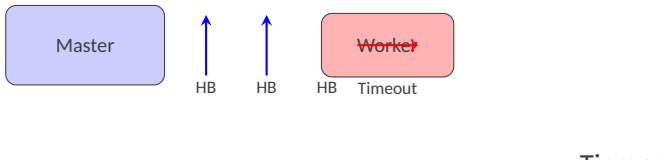
Tolerancia a Fallos

o



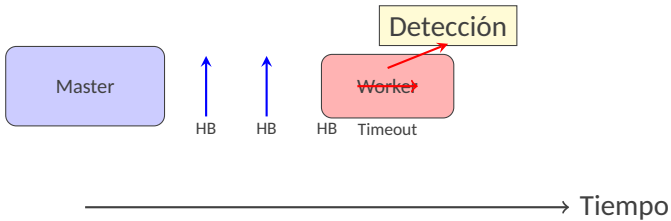
Tolerancia a Fallos

- o



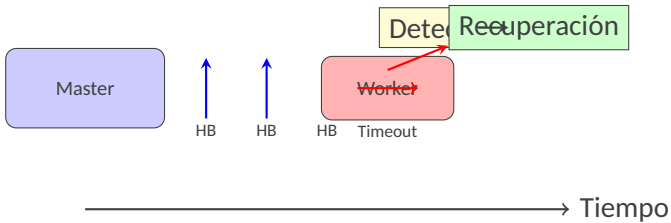
Tolerancia a Fallos

- o



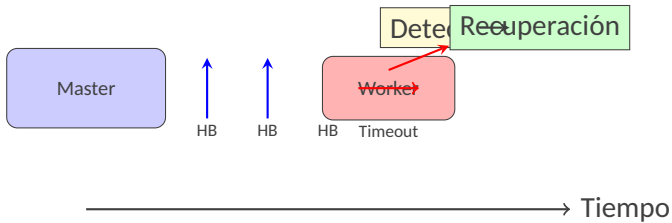
Tolerancia a Fallos

- o



Tolerancia a Fallos

o



Mecanismo de Heartbeat

- Intervalo: 3 segundos
- Timeout: 10 segundos
- Acción: Marcar nodo como caído

Demostración - Inicio del Cluster

o

```
$ ./start-java-cluster.sh
Starting Java distributed array cluster...
Starting master node on port 5000...
Master node PID: 12345
Starting worker-0...
Worker-0 PID: 12346
Starting worker-1...
Worker-1 PID: 12347
Starting worker-2...
Worker-2 PID: 12348

Java cluster started successfully!
Master node running on port 5000
3 worker nodes connected
```

Demostración - Cliente Interactivo

o

```
$ java -cp "out:lib/*" client.DistributedArrayClient localhost 5000
Connected to master at localhost:5000
Enter commands (type 'help' for usage, 'exit' to quit):
> create-double math-array 10000
Create array response: {"type":"OPERATION_COMPLETE",
  "data":{"arrayId":"math-array","status":"created"}}

> apply math-array example1
Apply operation response: {"type":"OPERATION_COMPLETE",
  "data":{"status":"processing"}}

> get math-array
Get result response: {"type":"OPERATION_COMPLETE",
  "data":{"status":"complete","result":"Operation completed"}}
```

Logs del Sistema

o

```
master.log
INFO: Master node started on port 5000
INFO: Worker registered: worker-0 from 127.0.0.1
INFO: Worker registered: worker-1 from 127.0.0.1
INFO: Worker registered: worker-2 from 127.0.0.1
INFO: Received array creation request: math-array (10000 elements)
INFO: Array segmented: 3 segments distributed
INFO: Processing operation: example1 on math-array

worker-0.log
INFO: Registered with master node
INFO: Received double array segment: math-array with 3333 elements
INFO: Processing Example 1 using 4 threads
INFO: Completed Example 1 processing for math-array
INFO: Sent result to master
```

Rendimiento y Escalabilidad

o

Paralelización

- Uso de todos los núcleos

Rendimiento y Escalabilidad

o

Paralelización

- Uso de todos los núcleos
- ThreadPool eficiente

Rendimiento y Escalabilidad

o

Paralelización

- Uso de todos los núcleos
- ThreadPool eficiente
- División automática de trabajo

Rendimiento y Escalabilidad

o

Paralelización

- Uso de todos los núcleos
- ThreadPool eficiente
- División automática de trabajo

Distribución

- Segmentación equitativa
- Comunicación asíncrona
- Procesamiento independiente

Rendimiento y Escalabilidad

o

Paralelización

- Uso de todos los núcleos
- ThreadPool eficiente
- División automática de trabajo

Distribución

- Segmentación equitativa
- Comunicación asíncrona
- Procesamiento independiente

Métricas (10,000 elementos)

- 1 worker: 250ms
- 2 workers: 140ms
- 3 workers: 95ms
- 4 workers: 75ms

Rendimiento y Escalabilidad

o

Paralelización

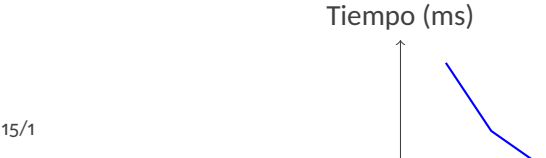
- Uso de todos los núcleos
- ThreadPool eficiente
- División automática de trabajo

Distribución

- Segmentación equitativa
- Comunicación asíncrona
- Procesamiento independiente

Métricas (10,000 elementos)

- 1 worker: 250ms
- 2 workers: 140ms
- 3 workers: 95ms
- 4 workers: 75ms



Conclusiones

o

Logros

- Librería funcional en Java y Python

Conclusiones

o

Logros

- Librería funcional en Java y Python
- Procesamiento verdaderamente distribuido

Conclusiones

o

Logros

- Librería funcional en Java y Python
- Procesamiento verdaderamente distribuido
- Paralelización efectiva por nodo

Conclusiones

o

Logros

- Librería funcional en Java y Python
- Procesamiento verdaderamente distribuido
- Paralelización efectiva por nodo
- Tolerancia a fallos básica implementada

Conclusiones

Logros

- Librería funcional en Java y Python
- Procesamiento verdaderamente distribuido
- Paralelización efectiva por nodo
- Tolerancia a fallos básica implementada
- Sin dependencias de frameworks externos

Conclusiones

o

Logros

- Librería funcional en Java y Python
- Procesamiento verdaderamente distribuido
- Paralelización efectiva por nodo
- Tolerancia a fallos básica implementada
- Sin dependencias de frameworks externos

Aplicaciones

- Procesamiento de grandes conjuntos de datos
- Cálculos científicos distribuidos
- Análisis de datos en paralelo

Conclusiones

o

Logros

- Librería funcional en Java y Python
- Procesamiento verdaderamente distribuido
- Paralelización efectiva por nodo
- Tolerancia a fallos básica implementada
- Sin dependencias de frameworks externos

Aplicaciones

- Procesamiento de grandes conjuntos de datos
- Cálculos científicos distribuidos
- Análisis de datos en paralelo

Mejoras Futuras

- Replicación activa completa

¿Preguntas?

o

Gracias por su atención

GitHub: `github.com/andre/distributed-array-lib`