Librería Distribuida para Arrays

Procesamiento Concurrente y Distribuido

CC4P1 Programación Concurrente y Distribuida

André Pacheco, Arbues Perez, Sergio Pezo

Julio 2025

FACULTAD DE CIENCIAS

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+
- TypeScript (Cliente)

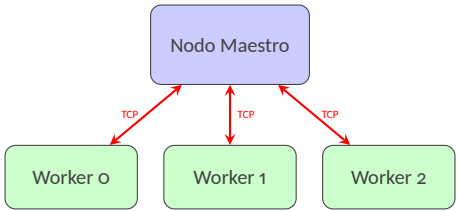
Arquitectura del Sistema

- o

Nodo Maestro

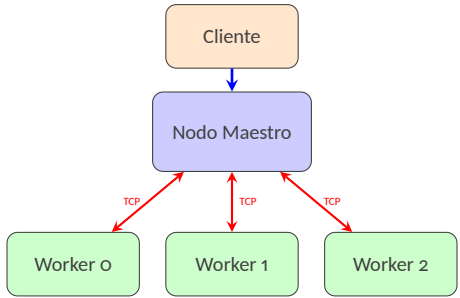
Arquitectura del Sistema

o



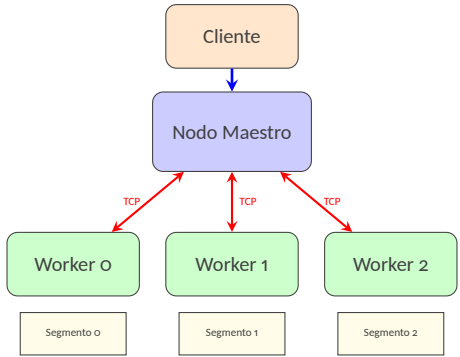
Arquitectura del Sistema

o



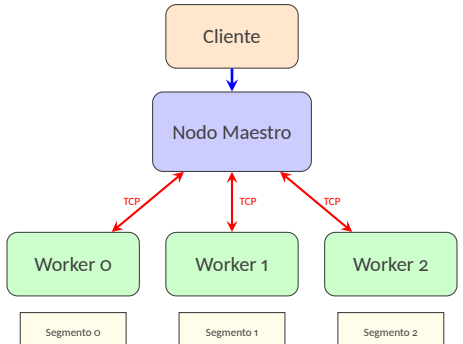
Arquitectura del Sistema

o



Arquitectura del Sistema

o



Características

- Arquitectura maestro-trabajador
- Distribución automática de datos
- Comunicación bidireccional

Implementación - Estructura

o

Java

- `MasterNode.java`

Python / TypeScript

- `master_node.py`

Implementación - Estructura

o

Java

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

Python / TypeScript

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

Implementación - Estructura

o

Java

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

Python / TypeScript

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

Implementación - Estructura

o

Java

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

Python / TypeScript

- `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 / TypeScript

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

Implementación - Estructura

o

Java

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

Python / TypeScript

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

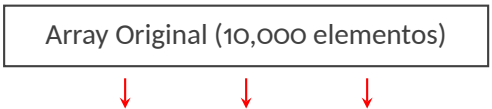
Segmentación de Arrays

o

Array Original (10,000 elementos)

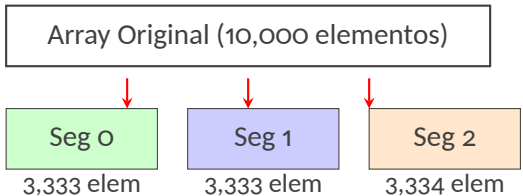
Segmentación de Arrays

- o



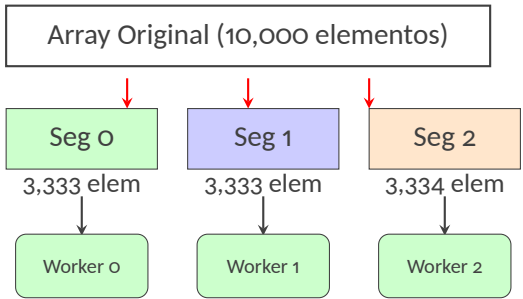
Segmentación de Arrays

o



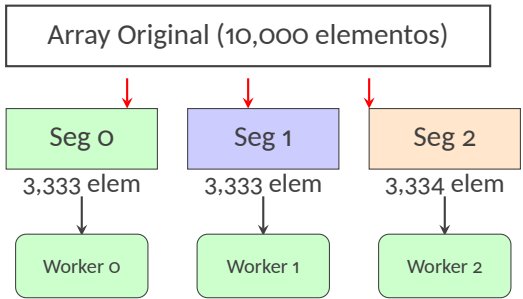
Segmentación de Arrays

o



Segmentación de Arrays

o



Algoritmo de Segmentación

- División equitativa: $\frac{\text{total}}{\text{workers}}$
- 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": {},
  "status": "OK"
}
```

Protocolo de Comunicación

o

Formato JSON

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

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": {},
  "status": "OK"
}
```

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": {},
  "status": "OK"
}
```

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": {},
  "status": "OK"
}
```

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": {},
  "status": "OK"
}
```

Tipos de Mensajes

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

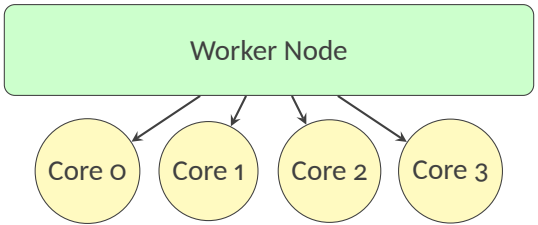
Procesamiento Paralelo

- o

Worker Node

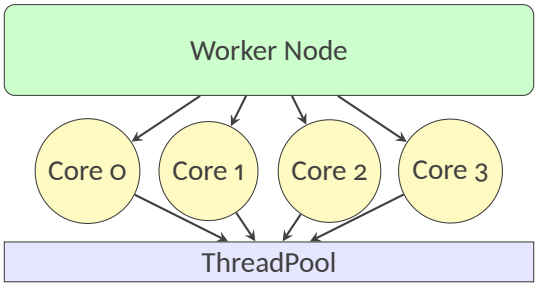
Procesamiento Paralelo

o



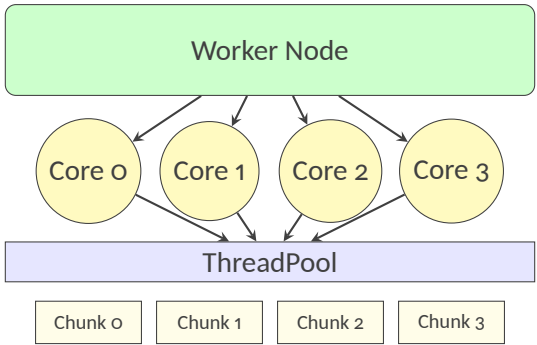
Procesamiento Paralelo

o



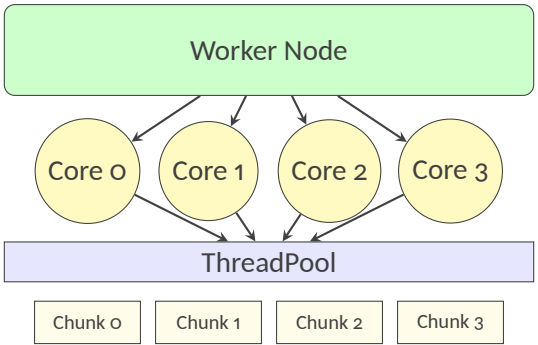
Procesamiento Paralelo

o



Procesamiento Paralelo

o



Estrategia

- Detección automática de núcleos: `Runtime.availableProcessors()`
- División del segmento en chunks
- Procesamiento concurrente con ThreadPool
- Sincronización mediante `Future<T>`

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

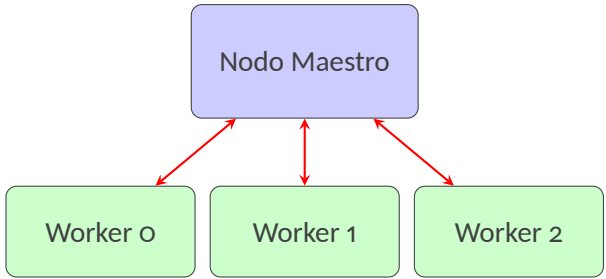
- Evaluación condicional para cada elemento
- Aplicación de transformación logarítmica
- Preservación de valores que no cumplen condición

Resiliencia

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

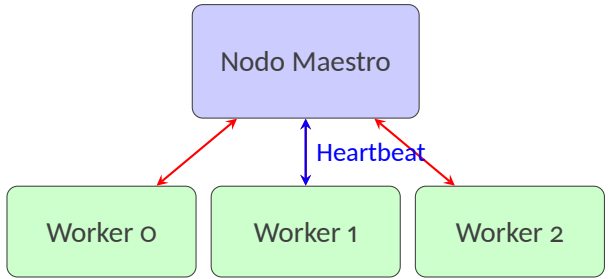
Tolerancia a Fallos

- o



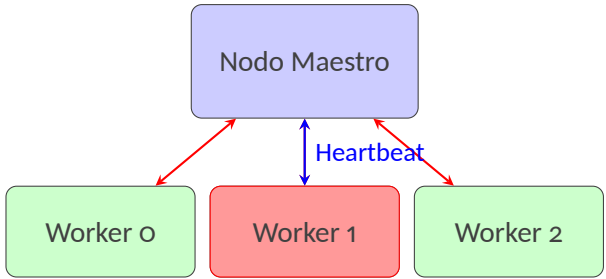
Tolerancia a Fallos

o



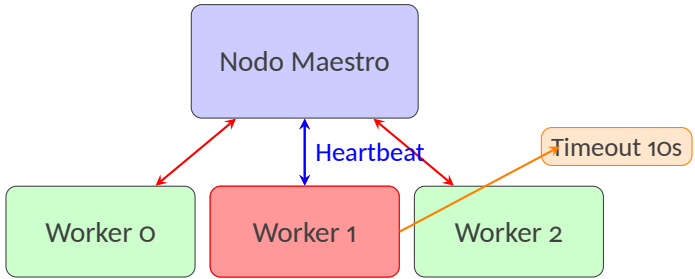
Tolerancia a Fallos

o



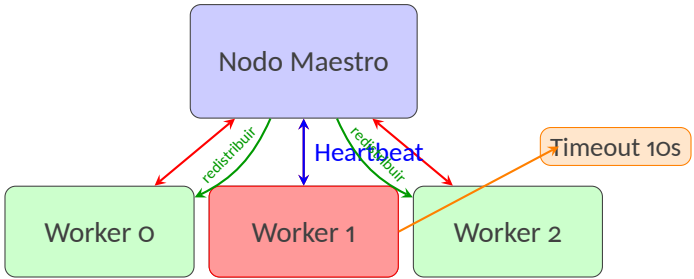
Tolerancia a Fallos

o



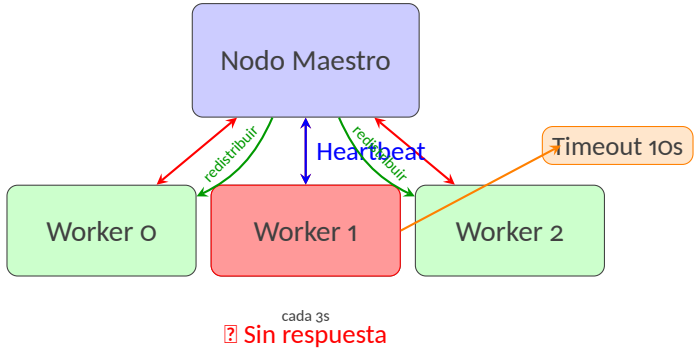
Tolerancia a Fallos

o



Tolerancia a Fallos

o



Mecanismo de Heartbeat

- Intervalo de verificación: 3 segundos
- Timeout de fallo: 10 segundos
- Acción: Redistribuir trabajo a nodos activos

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
```