



# Librería Distribuida para Arrays

Procesamiento Concurrente y Distribuido

CC4P1 Programación Concurrente y Distribuida

**André Pacheco, Arbues Perez, Sergio Pezo**

Julio 2025



# Agenda

0

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



## Objetivo del Proyecto

○

### Desarrollar una librería distribuida

- Arrays distribuidos: DArrayInt y DArrayDouble



# Objetivo del Proyecto

○

## Desarrollar una librería distribuida

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



## Objetivo del Proyecto

○

### Desarrollar una librería distribuida

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



# Objetivo del Proyecto

○

## Desarrollar una librería distribuida

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



# Objetivo del Proyecto

○

## 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

○

## 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

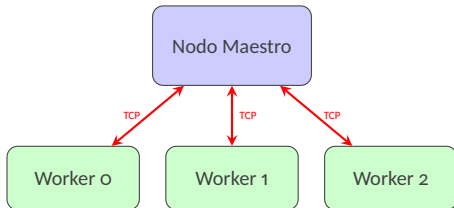
o

Nodo Maestro



# Arquitectura del Sistema

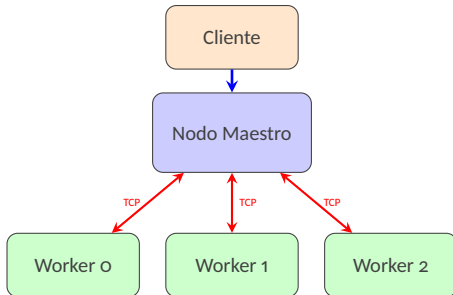
o





# Arquitectura del Sistema

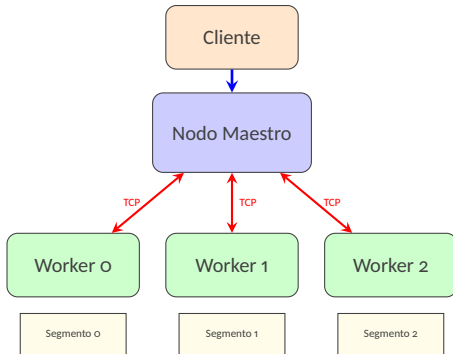
○





# Arquitectura del Sistema

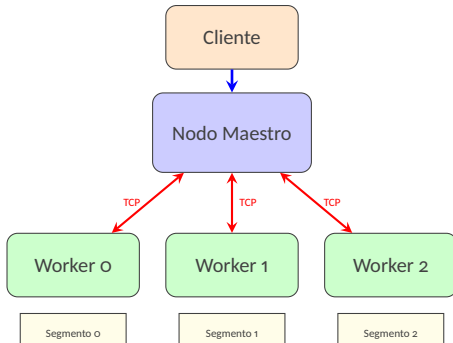
0





# Arquitectura del Sistema

0



## Características

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



# Implementación - Estructura

○

## Java

- `MasterNode.java`

## Python / TypeScript

- `master_node.py`



# Implementación - Estructura

○

## Java

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

## Python / TypeScript

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



# Implementación - Estructura

○

## Java

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

## Python / TypeScript

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





# Implementación - Estructura

○

## 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

○

## 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

○

## 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

○

Array Original (10,000 elementos)



# Segmentación de Arrays

○

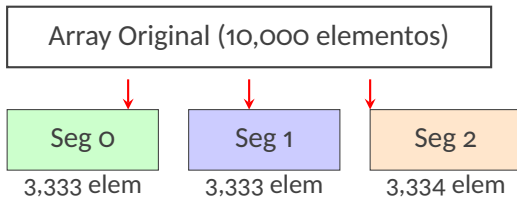
Array Original (10,000 elementos)





## Segmentación de Arrays

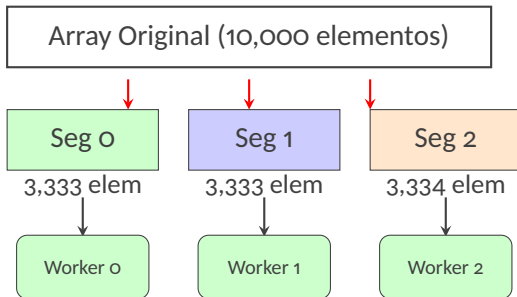
○





## Segmentación de Arrays

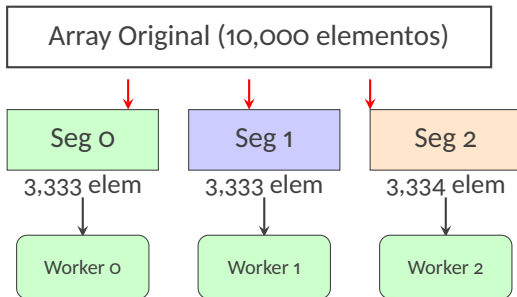
0





## Segmentación de Arrays

0



### 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
- REPLICATE\_DATA - Replicación de segmento



# 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
- REPLICATE\_DATA - Replicación de segmento
- RECOVER\_DATA - Recuperación de fallo



# Procesamiento Paralelo

○

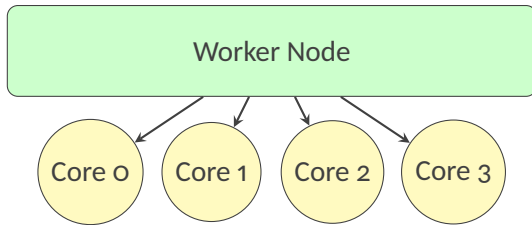
Worker Node





# Procesamiento Paralelo

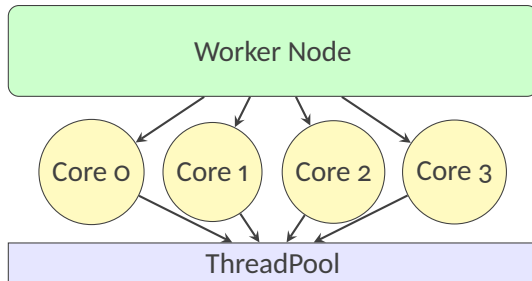
o





# Procesamiento Paralelo

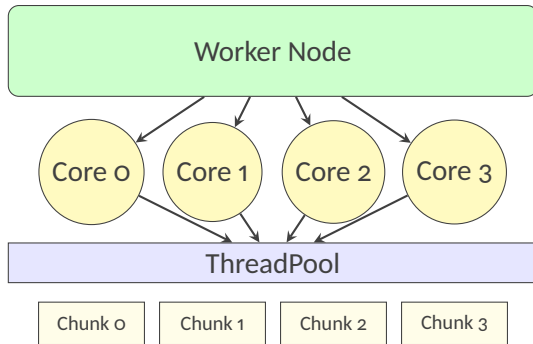
○





# Procesamiento Paralelo

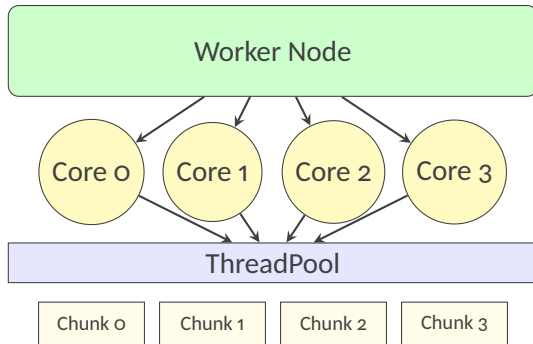
o





# Procesamiento Paralelo

0



## 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

○

### 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

0

### 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

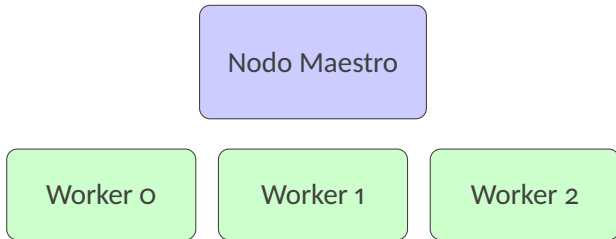
### Resiliencia

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



# Replicación de Datos

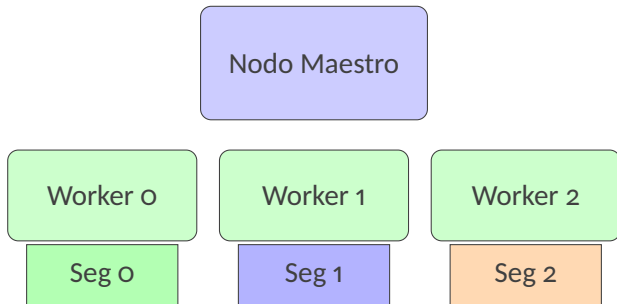
o





# Replicación de Datos

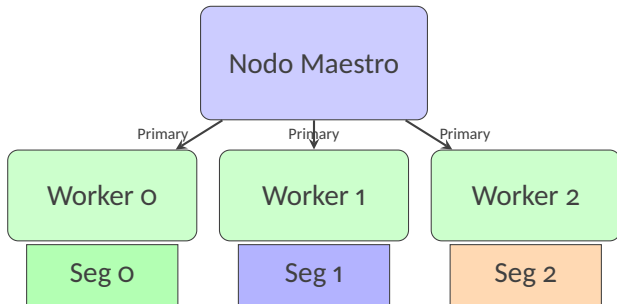
o





# Replicación de Datos

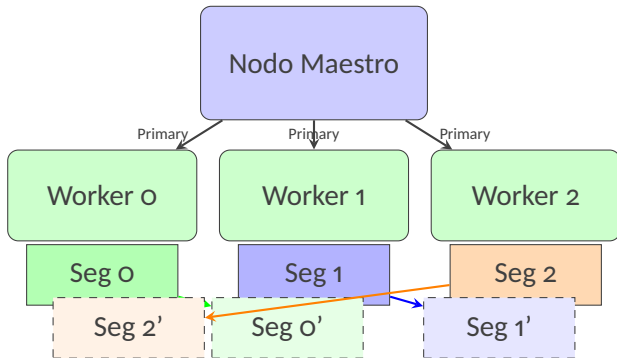
o





# Replicación de Datos

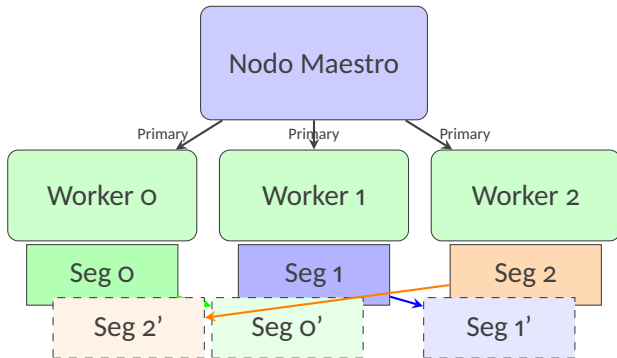
o





# Replicación de Datos

0



Factor de replicación = 2 (primario + 1 réplica)



# Mecanismo de Recuperación

o

## Detección de Fallos

1. Heartbeat timeout (10s)





# Mecanismo de Recuperación

o

## Detección de Fallos

1. Heartbeat timeout (10s)
2. Worker marcado como caído



# Mecanismo de Recuperación

o

## Detección de Fallos

1. Heartbeat timeout (10s)
2. Worker marcado como caído
3. Activar proceso de recuperación



# Mecanismo de Recuperación

o

## Detección de Fallos

1. Heartbeat timeout (10s)
2. Worker marcado como caído
3. Activar proceso de recuperación

## Promoción de Réplicas

1. Identificar segmentos afectados
2. Promover réplicas a primarias
3. Actualizar mapeos de segmentos



# Mecanismo de Recuperación

O

## Detección de Fallos

1. Heartbeat timeout (10s)
2. Worker marcado como caído
3. Activar proceso de recuperación

## Creación de Nuevas Réplicas

1. Seleccionar workers disponibles
2. Replicar datos desde primario
3. Mantener factor de replicación

## Promoción de Réplicas

1. Identificar segmentos afectados
2. Promover réplicas a primarias
3. Actualizar mapeos de segmentos



# Mecanismo de Recuperación

0

## Detección de Fallos

1. Heartbeat timeout (10s)
2. Worker marcado como caído
3. Activar proceso de recuperación

## Creación de Nuevas Réplicas

1. Seleccionar workers disponibles
2. Replicar datos desde primario
3. Mantener factor de replicación

## Promoción de Réplicas

1. Identificar segmentos afectados
2. Promover réplicas a primarias
3. Actualizar mapeos de segmentos

## Redistribución

1. Balancear carga entre workers
2. Evitar sobrecarga de nodos
3. Optimizar uso de recursos



## Ejemplo 3: Recuperación ante Fallos

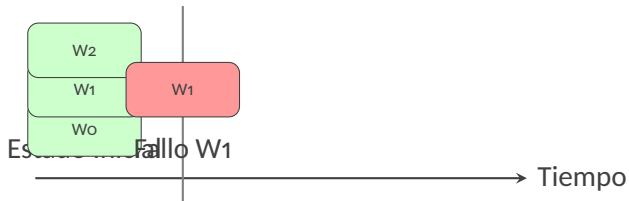
o





## Ejemplo 3: Recuperación ante Fallos

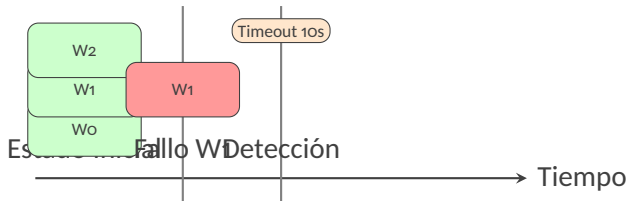
○





## Ejemplo 3: Recuperación ante Fallos

o

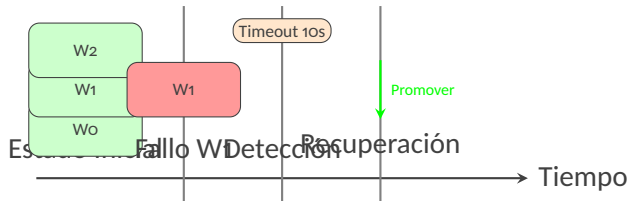






## Ejemplo 3: Recuperación ante Fallos

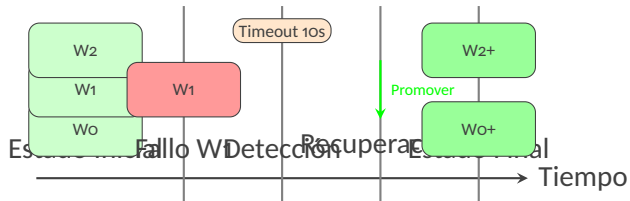
○





## Ejemplo 3: Recuperación ante Fallos

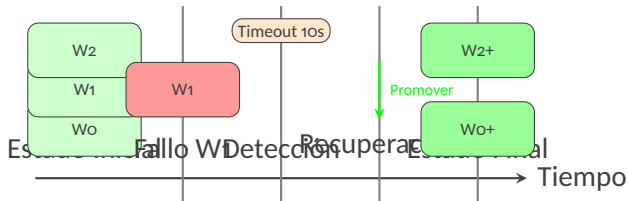
○





## Ejemplo 3: Recuperación ante Fallos

0



### Proceso Automático

- Sin pérdida de datos
- Continuidad del servicio
- Transparente para el cliente



# Demostración - Recuperación Automática

0

```
$ ./test-recovery.sh
=== Distributed Array Recovery Test ===
Starting Master node on port 5000
Starting Worker-1
Starting Worker-2
Starting Worker-3

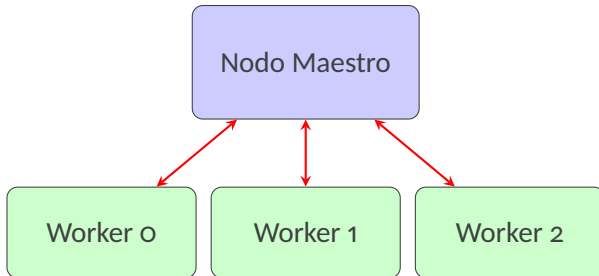
=== Creating distributed array ===
Create array response: {status:created,arrayId:myArray}
INFO: Replicated segment 0 to worker-2
INFO: Replicated segment 100 to worker-3
INFO: Replicated segment 200 to worker-1

=== Simulating Worker-2 failure ===
Worker-2 has been terminated!
WARNING: Worker worker-2 failed health check
ERROR: Handling failure of worker: worker-2
```



# Tolerancia a Fallos

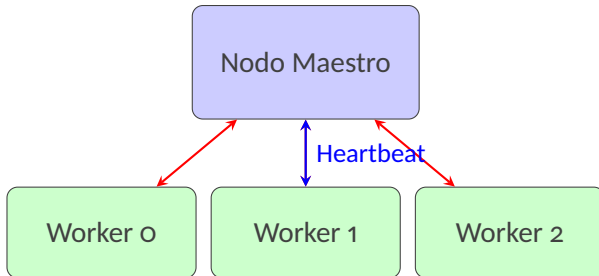
0





# Tolerancia a Fallos

0

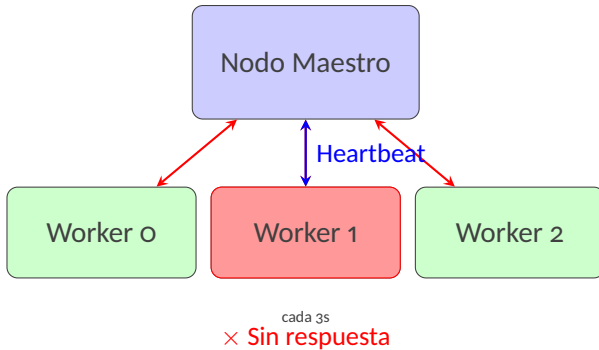


cada 3s



# Tolerancia a Fallos

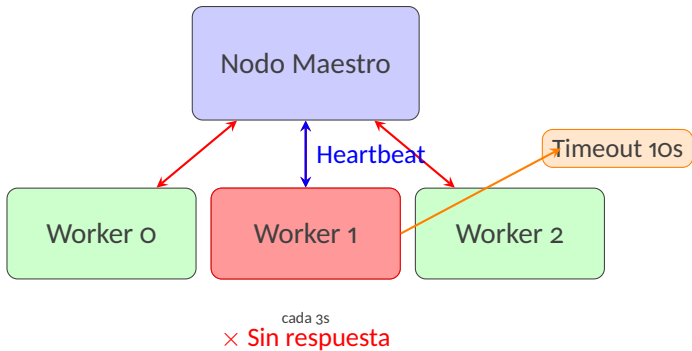
0





# Tolerancia a Fallos

0

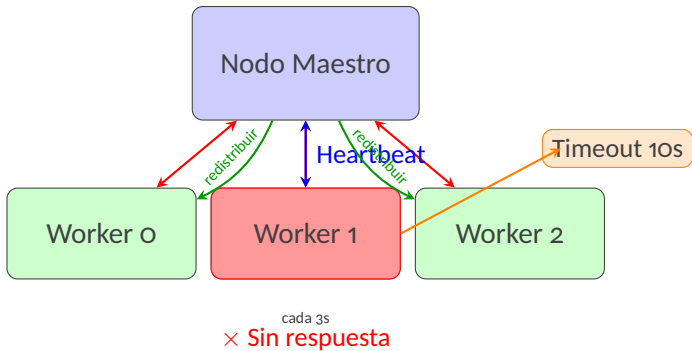






# Tolerancia a Fallos

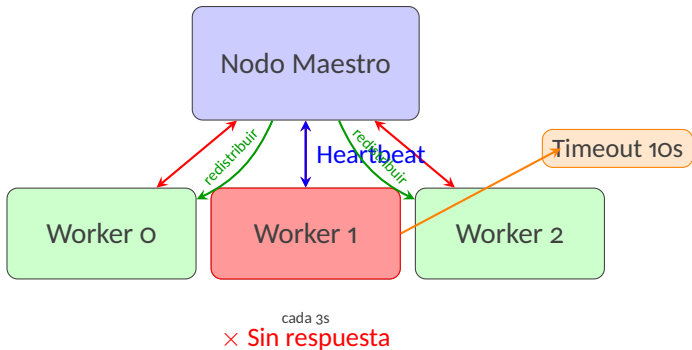
0





# Tolerancia a Fallos

0



## Sistema de Tolerancia a Fallos

- Heartbeat: verificación cada 3 segundos
- Detección: timeout de 10 segundos

15/28 Replicación: factor 2 (primario + réplica)



## Demostración - Inicio del Cluster

0

```
$ ./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
```



# Cliente TypeScript

○

## Características

- Cliente completo en TypeScript/Node.js



# Cliente TypeScript

○

## Características

- Cliente completo en TypeScript/Node.js
- Compatible con clusters Java y Python



# Cliente TypeScript

○

## Características

- Cliente completo en TypeScript/Node.js
- Compatible con clusters Java y Python
- Interfaz CLI idéntica



# Cliente TypeScript

o

## Características

- Cliente completo en TypeScript/Node.js
- Compatible con clusters Java y Python
- Interfaz CLI idéntica
- Comunicación asíncrona con Promises



# Cliente TypeScript

○

## Características

- Cliente completo en TypeScript/Node.js
- Compatible con clusters Java y Python
- Interfaz CLI idéntica
- Comunicación asíncrona con Promises
- Tipado fuerte con interfaces





# Cliente TypeScript

0

## Características

- Cliente completo en TypeScript/Node.js
- Compatible con clusters Java y Python
- Interfaz CLI idéntica
- Comunicación asíncrona con Promises
- Tipado fuerte con interfaces

## Ejemplo de uso

```
$ npm start -- localhost 5000
Connected to master at localhost:5000
Enter commands (type help for usage, exit to quit):
> create-double ts-array 5000
Create array response: {status:created}
> apply ts-array example1
Apply operation response: {status:processing}
```



## Demostración - Cliente Interactivo

0

```
$ 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

0

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

○

## 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

0

## Paralelización

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

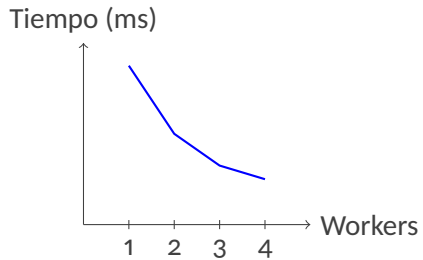
## Distribución

- Segmentación equitativa
- Comunicación asíncrona
- Procesamiento independiente

## Métricas (10,000 elementos)

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







# Conclusiones

0

## Logros

- Librería funcional en Java, Python y TypeScript



# Conclusiones

0

## Logros

- Librería funcional en Java, Python y TypeScript
- Procesamiento verdaderamente distribuido



# Conclusiones

0

## Logros

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



# Conclusiones

0

## Logros

- Librería funcional en Java, Python y TypeScript
- Procesamiento verdaderamente distribuido
- Paralelización efectiva por nodo
- Sistema completo de replicación y recuperación



# Conclusiones

O

## Logros

- Librería funcional en Java, Python y TypeScript
- Procesamiento verdaderamente distribuido
- Paralelización efectiva por nodo
- Sistema completo de replicación y recuperación
- Sin dependencias de frameworks externos



# Conclusiones

0

## Logros

- Librería funcional en Java, Python y TypeScript
- Procesamiento verdaderamente distribuido
- Paralelización efectiva por nodo
- Sistema completo de replicación y recuperación
- Sin dependencias de frameworks externos
- Interoperabilidad entre lenguajes

## Aplicaciones

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



# Conclusiones

0

## Logros

- Librería funcional en Java, Python y TypeScript
- Procesamiento verdaderamente distribuido
- Paralelización efectiva por nodo
- Sistema completo de replicación y recuperación
- Sin dependencias de frameworks externos
- Interoperabilidad entre lenguajes

## Aplicaciones

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





## Preguntas

o

Gracias por su atención

GitHub: <https://github.com/A-PachecoT/distributed-array-lib>

