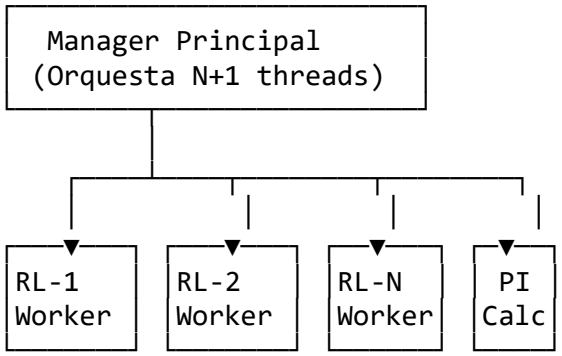
1. DISEÑO CONCISO: REGRESIÓN LINEAL CON CONCURRENCIA Y PI

Javier Felipe Rosero Sandoval

Resumen

Paralelizar regresión lineal en N threads + 1 thread calculando PI en paralelo usando Monte Carlo.

Arquitectura



Componentes Principales

1. DataPartitioner

- Divide dataset en N mini-batches iguales
- Valida cobertura completa
- Thread-safe con mutex

Métodos:

```
create_partitions()          → divide en N partes
get_partition(id)            → retorna partición i
validate_partitions()        → verifica integridad
```

2. LinearRegressionWorker (×N threads)

- Procesa mini-batch asignado
- Calcula gradientes locales
- Sincroniza con barrera cada `sync_frequency` épocas

Métodos:

```
compute_gradient()           → (dw, db, mse)
update_weights(dw, db)       → actualiza parámetros
```

```
synchronize_with_others()    → promedia con otros workers
run(epochs)                  → loop principal
```

Sincronización: - Cada worker calcula independientemente - Cada `sync_frequency` épocas: promedian pesos - Barrera sincroniza a todos antes de continuar - Ejemplo: epochs 0-49 independientes → epoch 50 sincronización

3. MonteCarloPiCalculator (1 thread)

- Genera puntos aleatorios en $[0,1]^2$
- Cuenta cuántos caen dentro círculo unitario
- PI ≈ 4 × (dentro / total)

Métodos:

```
generate_random_points(n)    → genera n puntos
count_inside_circle()        → cuenta puntos dentro
estimate_pi()                → calcula π estimado
run_concurrent()             → emite estimaciones cada 100ms
```

4. ConcurrentLinearRegressionManager

- Crea ThreadPoolExecutor con N+1 threads
- Lanza workers en paralelo
- Sincroniza con CyclicBarrier
- Promedia resultados finales

Métodos:

```
initialize(X, y, num_threads)
train_async(epochs)          → retorna Future
predict(x)                   → predicción thread-safe
get_metrics()                → histórico convergencia
```

Flujo de Ejecución

```
T0: Inicialización
├─ Particionar datos en 2 workers
├─ W1: [1,2,3] → y=[2,4,6]
└─ W2: [4,5] → y=[8,10]

T1-T50: Épocas 1-50 (sin sincronización)
├─ W1: Gradiente descent local
├─ W2: Gradiente descent local
├─ PI: Monte Carlo paralelo
└─ DIVERGEN (distintos w, b locales)

T51: Sincronización @ epoch 50
├─ W1: w=1.2, b=0.15
├─ W2: w=1.8, b=0.25
```

```
├─ Promedio: w=1.5, b=0.2
├─ AMBOS actualizan a promedios
└─ Barrera: esperan juntos

T52-T100: Épocas 51-100
├─ Ambos con w=1.5, b=0.2 iniciales
├─ Vuelven a divergir
└─ Convergen gradualmente

T101: Final
├─ w_final ≈ 2.0
├─ b_final ≈ 0.0
└─ PI ≈ 3.14159265
```

Ventajas

Ventaja	Descripción
Escalabilidad	O(1) overhead con N threads
CPU 100%	N+1 threads siempre activos
Monitoreo vivo	PI se estima en tiempo real
Convergencia	Sincronización estabiliza aprendizaje
Thread-safe	Mutex + Barrier previenen races

Consideraciones

Overhead de sincronización: Cada barrera tiene costo (~1-2ms)

Divergencia temporal: Workers divergen entre sincronizaciones (normal)
Comunicación: Usar memoria compartida, no paso de mensajes
