

Sistema Distribuido para Entrenamiento de IAs

con Algoritmo de Consenso RAFT

Espinoza Valera, Cinver Alem
Mercado Taype, Ariana Aracely
Cadillo Tarazona, Jharvy Jonas
Centeno Leon, Martin Alonso
Calapuja Apaza, Luis Alberto

Universidad Nacional de Ingeniería
Escuela de Ciencias de la Computación

CC4P1 - Programación Concurrente y Distribuida

Final 2025-I

Agenda de Exposición

Cliente Python

Cinver Alem Espinoza

Red TCP y Monitoreo

Martin Alonso Centeno

Algoritmo RAFT

Luis Alberto Calapuja



Ingeniería de Datos

Ariana Aracely Mercado

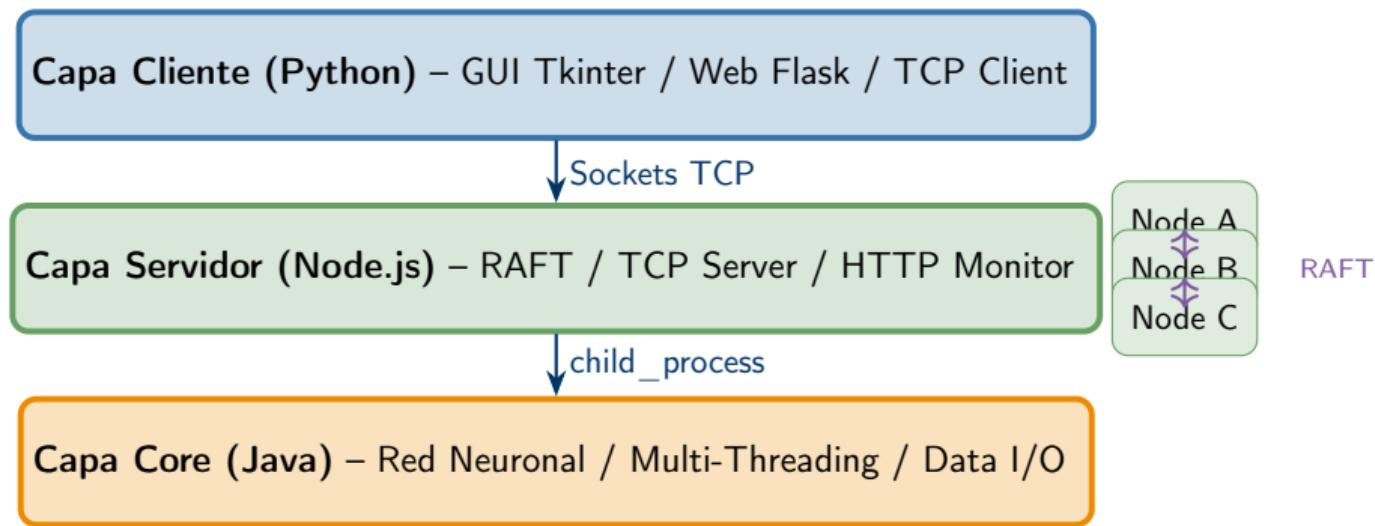


Motor de IA

Jharvy Jonas Cadillo

Arquitectura: Python (Cliente) → Node.js (Servidor/RAFT) → Java (Core IA)

Arquitectura del Sistema



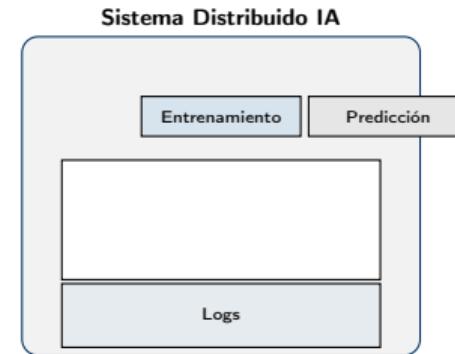
Cinver Alem Espinoza

Cliente Python (Frontend)

GUI Desktop, Web App y Cliente TCP

Características:

- Interfaz con pestañas (Notebook)
- Tab **Entrenamiento**: Subir CSV y nombrar modelo
- Tab **Predicción**: Seleccionar modelo y enviar datos
- Área de **logs** en tiempo real
- Ejecución en threads separados (no bloquea UI)



Rutas implementadas:

- GET / – Página principal (HTML)
- POST /train – Enviar dataset
- POST /predict – Solicitar predicción

Ventajas:

- Accesible desde cualquier navegador
- Upload de archivos con FormData
- Interfaz responsive (Bootstrap)

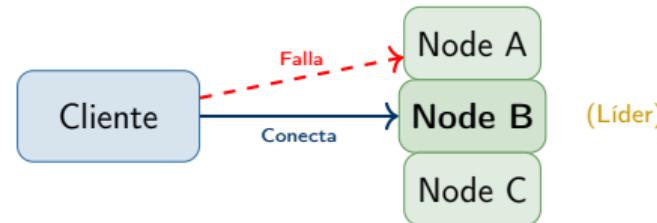
```
@app.route('/train', methods=['POST'])
def train():
    model_name = request.form.get(
        'model_name')
    file = request.files.get('file')
    file_bytes = file.read()

    response = client.
        train_model_from_bytes(
            model_name, file_bytes
        )
    return jsonify(response)
```

Cliente TCP - Conexión al Clúster

Características principales:

- Conexión a **cualquier nodo** del clúster (round-robin)
- **Reconexión automática** si el líder cambia
- Serialización de mensajes en **JSON** + **delimitador \n**
- Codificación de archivos en **Base64**



Script de Pruebas de Estrés

Objetivo

Validar consistencia del sistema enviando **1000+ peticiones aleatorias**

```
for i in range(NUM_REQUESTS):    # 1000
    req_type = random.choice(['TRAIN', 'PREDICT'])

    if req_type == 'TRAIN':
        resp = client.train_model(f"stress_model_{i}", TEST_FILE)
    else:
        input_vector = [random.random() for _ in range(4)]
        resp = client.predict("stress_model_0", input_vector)
```

Métricas reportadas: Peticiones exitosas, fallidas, tiempo total, req/segundo

Martin Alonso Centeno

Red TCP y Monitoreo

Servidores TCP, HTTP Monitor, Integración Java

Servidor TCP para Clientes

Implementación:

- Módulo nativo `net` de Node.js
- **Sin** WebSockets ni `Socket.io`
- Buffer para mensajes fragmentados
- Delimitador `\n`

Puertos:

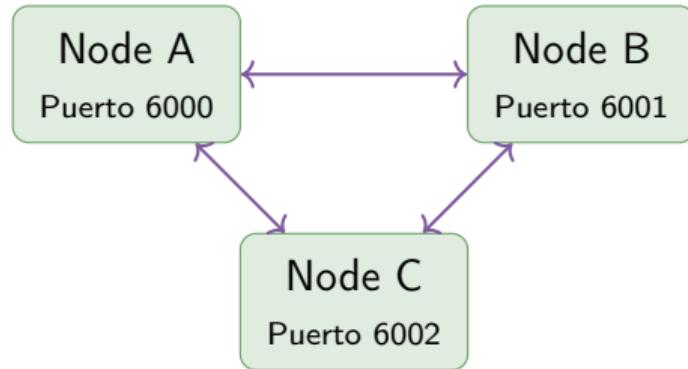
- Node A: 5000
- Node B: 5001
- Node C: 5002

```
const server = net.createServer((socket)
  => {
    socket.on('data', (data) => {
      buffer += data.toString();

      while ((idx = buffer.indexOf('\n')) !== -1) {
        const raw = buffer.slice(0,
          idx);
        buffer = buffer.slice(idx + 1)
        ;
        const msg = JSON.parse(raw);
        messageHandler(socket, msg);
      }
    });
});
```

Servidor TCP para Peers (Nodos)

Comunicación entre nodos del clúster:



Funciones:

- `connectToPeer()` – Establece conexión con reintentos
- `sendToPeer(peerId, msg)` – Envía a un nodo específico
- `broadcastToPeers(msg)` – Envía a todos los peers

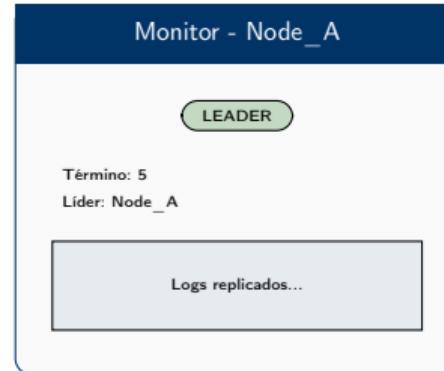
Monitor HTTP - Panel Web

Información mostrada:

- Rol actual (Leader/Follower/Candidate)
- Término RAFT actual
- Líder conocido
- Archivos en disco
- Log de operaciones

Puertos HTTP:

- Node A: <http://localhost:8080>
- Node B: <http://localhost:8081>
- Node C: <http://localhost:8082>



Integración con Java (child_process)

Ejecución del Core IA desde Node.js:

```
const { spawn } = require('child_process');

function trainModel(inputPath, modelId) {
    return new Promise((resolve, reject) => {
        const process = spawn('java', [
            '-jar', JAR_PATH,
            'train', inputPath, modelId
        ]);

        process.stdout.on('data', (data) => {
            console.log('[JAVA] ${data}');
        });

        process.on('close', (code) => {
            if (code === 0) resolve({ success: true, modelId });
            else reject(new Error(`Exit code: ${code}`));
        });
    });
}
```

Configuración de Nodos

```
// config.js
const allNodes = {
  'Node_A': { host: '127.0.0.1', clientPort: 5000,
               peerPort: 6000, httpPort: 8080 },
  'Node_B': { host: '127.0.0.1', clientPort: 5001,
               peerPort: 6001, httpPort: 8081 },
  'Node_C': { host: '127.0.0.1', clientPort: 5002,
               peerPort: 6002, httpPort: 8082 }
};

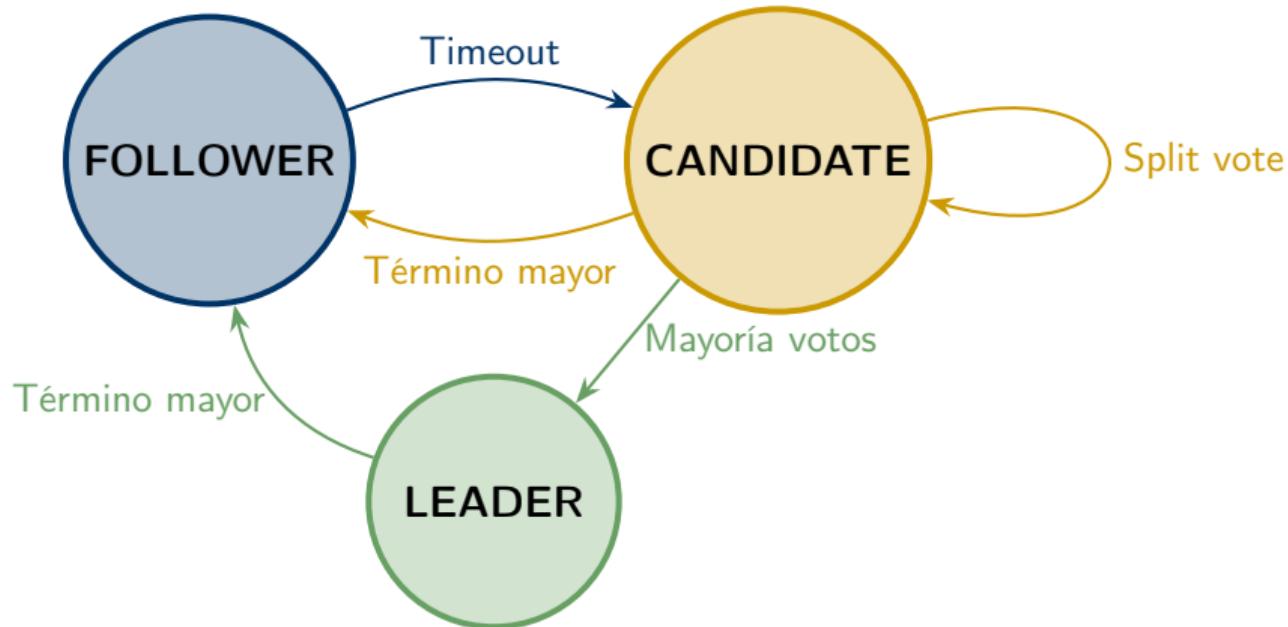
// Obtener peers (todos menos yo)
const peers = Object.entries(allNodes)
  .filter(([id]) => id !== nodeId)
  .map(([id, config]) => ({ id, host: config.host,
                               port: config.peerPort }));
```

Luis Alberto Calapuja

Algoritmo RAFT

Consenso Distribuido y Replicación

Estados del Nodo RAFT



Election Timeout y Votación

Proceso de elección:

- ① Timeout aleatorio (150-300ms) expira sin heartbeat
- ② Nodo incrementa término y se vuelve **Candidate**
- ③ Vota por sí mismo y envía REQUEST_VOTE
- ④ Espera respuestas hasta obtener **quórum** (mayoría)

```
startElection() {
    this.currentTerm++;
    this.votedFor = config.nodeId;
    this.votesReceived = 1; // Voto por mi mismo

    broadcastToPeers({
        type: 'REQUEST_VOTE',
        term: this.currentTerm,
        candidateId: config.nodeId,
        lastLogIndex: this.log.length - 1
    });
}
```

Heartbeats y AppendEntries

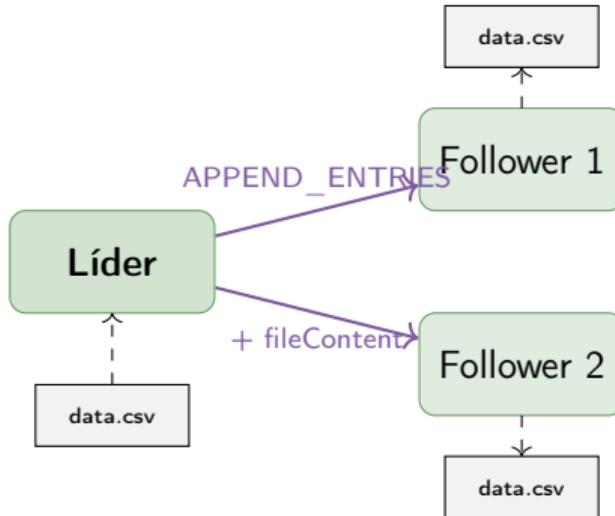
Líder envía heartbeats:

- Cada 50ms
- Mantiene autoridad
- Evita nuevas elecciones
- Replica logs a followers

```
sendHeartbeats() {
    if (this.state !== STATES.LEADER)
        return;

    broadcastToPeers({
        type: 'APPEND_ENTRIES',
        term: this.currentTerm,
        leaderId: config.nodeId,
        entries: [],
        leaderCommit: this.commitIndex
    });
}
```

Replicación de Datos



El archivo se replica a todos los nodos antes de confirmar al cliente

Estructura del Log RAFT

```
// Entrada en el log
{
    index: 5,
    term: 3,
    command: 'STORE_FILE',
    filename: 'dataset_v1.csv',
    timestamp: 1702500000000
}

// Persistencia en disco
fs.writeFileSync(
    path.join(LOGS_PATH, 'raft_log.json'),
    JSON.stringify(logEntries, null, 2)
);
```

Carpeta /disk: /logs (historial RAFT), /datasets (archivos), /models (modelos)

Ariana Aracely Mercado

Ingeniería de Datos

DataLoader, Normalización y Gestión I/O

DataLoader - Carga de Datos

Funcionalidades:

- Lectura de archivos **CSV**
- Detección automática de **headers**
- Separación de **features** y **labels**
- Manejo de errores por línea

```
// Formato CSV: feature1,feature2,...,label
while ((line = br.readLine()) != null) {
    String[] parts = line.split(",");
    // Features: todas menos la ultima
    double[] input = new double[parts.length - 1];
    for (int i = 0; i < parts.length - 1; i++)
        input[i] = Double.parseDouble(parts[i]);
    // Label: ultima columna
    double[] output = {Double.parseDouble(parts[parts.length-1])};
}
```

Normalización Min-Max

Fórmula

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \rightarrow \text{Escala valores a } [0, 1]$$

```
private double[][] normalizeData(double[][] data) {
    // Encontrar min y max por feature
    for (double[] row : data) {
        for (int i = 0; i < features; i++) {
            if (row[i] < min[i]) min[i] = row[i];
            if (row[i] > max[i]) max[i] = row[i];
        }
    }
    // Normalizar
    for (int i = 0; i < data.length; i++) {
        for (int j = 0; j < features; j++) {
            normalized[i][j] = (data[i][j] - min[j]) / (max[j] - min[j]);
        }
    }
    return normalized;
}
```

TrainingData - Estructura de Datos

Clase para almacenar ejemplos de entrenamiento:

```
public class TrainingData {  
    private final double[][] inputs;  
    private final double[][] outputs;  
  
    // Dividir en batches para procesamiento paralelo  
    public TrainingData[] splitIntoBatches(int batchSize) {  
        int numBatches = (int) Math.ceil((double) getSize() / batchSize);  
        // ...crear batches  
    }  
  
    // Mezclar datos (shuffle) para cada epoch  
    public void shuffle() {  
        Random rand = new Random();  
        for (int i = inputs.length - 1; i > 0; i--) {  
            int j = rand.nextInt(i + 1);  
            // swap inputs[i] <-> inputs[j]  
        }  
    }  
}
```

Parseo de Vectores de Entrada

Para predicciones desde el cliente:

```
// Soporta multiples formatos:  
// "0.5,0.1,0.9"  
// "[0.5, 0.1, 0.9]"  
// "0.5, 0.1, 0.9"  
  
public double[] parseInputVector(String inputData) {  
    inputData = inputData.trim()  
        .replace("[", "")  
        .replace("]", "")  
        .replace(" ", "");  
  
    String[] parts = inputData.split(",");  
    double[] vector = new double[parts.length];  
  
    for (int i = 0; i < parts.length; i++)  
        vector[i] = Double.parseDouble(parts[i]);  
  
    return vector;  
}
```

Generador de Datos de Prueba

Datasets sintéticos para testing:

uniblue!20	Tipo	Ejemplos	Descripción
	xor	4	Problema XOR clásico
	linear	100	Relación lineal $y = 2x + 1$
	circles	500	Círculos concéntricos
	large	10,000	Pruebas de estrés

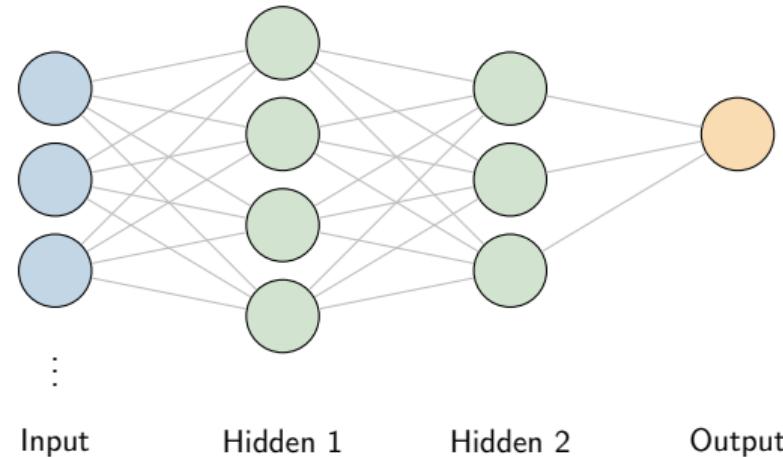
Uso: `java -cp bin DataGenerator xor`

Jharvy Jonas Cadillo

Motor de IA

Red Neuronal y Multi-Threading

Arquitectura de la Red Neuronal



Configuración: int[] layers = {inputSize, 64, 32, outputSize}

Forward Propagation

Algoritmo

Para cada capa: $z = W \cdot a + b$ luego $a = \sigma(z)$

```
public double[] predict(double[] input) {
    Matrix activation = new Matrix(input);

    for (int i = 0; i < weights.length; i++) {
        Matrix z = weights[i].multiply(activation).add(biases[i]);

        if (i < weights.length - 1)
            activation = ActivationFunction.applyReLU(z);      // Ocultas
        else
            activation = ActivationFunction.applySigmoid(z); // Salida
    }
    return activation.toArray();
}
```

Backpropagation

Algoritmo de entrenamiento:

- ① Forward pass (guardar activaciones)
- ② Calcular error: $\delta = (output - target) \odot \sigma'(z)$
- ③ Propagar hacia atrás: $\delta^{(l)} = (W^{(l+1)})^T \delta^{(l+1)} \odot \sigma'(z^{(l)})$
- ④ Actualizar pesos: $W = W - \eta \cdot \nabla W$

```
// Actualizar pesos y biases
for (int i = 0; i < weights.length; i++) {
    weights[i] = weights[i].subtract(
        weightGradients[i].scale(learningRate)
    );
    biases[i] = biases[i].subtract(
        biasGradients[i].scale(learningRate)
    );
}
```

Multi-Threading con ExecutorService

```
public class MultiThreadTrainer {  
    private final int numThreads;  
    private final ExecutorService executor;  
  
    public MultiThreadTrainer(NeuralNetwork network) {  
        // Usar TODOS los nucleos disponibles  
        this.numThreads = Runtime.getRuntime().availableProcessors();  
        this.executor = Executors.newFixedThreadPool(numThreads);  
    }  
  
    // Entrenar batches en paralelo  
    for (TrainingData batch : batches) {  
        executor.submit(() -> {  
            double batchLoss = trainSingleBatch(batch, learningRate);  
            totalLoss.add(batchLoss);  
            latch.countDown();  
        });  
    }  
    latch.await(); // Esperar todos los threads  
}
```

Serialización de Modelos

Guardar y cargar modelos entrenados:

```
// Guardar modelo
public void saveModel(String filePath) throws IOException {
    try (ObjectOutputStream oos = new ObjectOutputStream(
        new BufferedOutputStream(new FileOutputStream(filePath))) {
        oos.writeObject(this);
    }
}

// Cargar modelo
public static NeuralNetwork loadModel(String filePath) {
    try (ObjectInputStream ois = new ObjectInputStream(
        new BufferedInputStream(new FileInputStream(filePath))) {
        return (NeuralNetwork) ois.readObject();
    }
}
```

Formato: Binario Java (.bin) - Serialización nativa



Demo en Vivo

- ① Iniciar clúster de 3 nodos
- ② Observar elección de líder
- ③ Entrenar modelo desde cliente
- ④ Verificar replicación en monitores
- ⑤ Realizar predicción
- ⑥ Simular caída de líder

Conclusiones

- ✓ Sistema distribuido funcional con **3 lenguajes** de programación
- ✓ Algoritmo **RAFT** garantiza consistencia y tolerancia a fallos
- ✓ Red neuronal implementada **desde cero** sin frameworks
- ✓ **Multi-threading** aprovecha todos los núcleos del CPU
- ✓ Comunicación mediante sockets **TCP nativos**
- ✓ Arquitectura **escalable** y modular

¿Preguntas?

¡Gracias por su atención!

CC4P1 - Programación Concurrente y Distribuida
Final 2025-I