**UNIVERSIDAD TECNOLÓGICA DE SANTIAGO, UTESA**

**SISTEMA CORPORATIVO**

**FACULTAD DE INGENIERÍA Y ARQUITECTURA**

**CARRERA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES**

**A green and white circle with white text

AI-generated content may be incorrect.**

**Algoritmos paralelos**

Tarea semana 2

**Instructor:**

Ma. Iván Mendoza

**Presentado Por:**

Malvin Omar Almonte Almonte           1-21-2391

10 de septiembre del 2025

**Santiago de los Caballeros, República Dominicana**

**Árboles balanceados**

Los árboles balanceados son estructuras de datos que buscan mantener un equilibrio entre sus ramas para que las operaciones de búsqueda, inserción y eliminación sean rápidas. La idea es que ninguna rama sea demasiado larga en comparación con las otras, lo que asegura que podamos encontrar elementos de manera eficiente. Ejemplos comunes son los árboles AVL, Red-Black y B-Tree, y su principal ventaja es que permiten búsquedas en tiempo logarítmico.

**Técnica Pointer Jumping**

Pointer Jumping es una técnica usada principalmente en algoritmos paralelos para recorrer estructuras enlazadas, como listas o árboles. En vez de moverse nodo por nodo, cada nodo salta al siguiente nodo de un nivel superior, acelerando el recorrido. Esto reduce considerablemente el tiempo de ejecución y es útil cuando se trabaja con procesamiento distribuido o paralelo.

**Divide y vencerás**

Divide y vencerás es una estrategia clásica de programación que consiste en dividir un problema grande en subproblemas más pequeños, resolver cada uno por separado y luego combinar los resultados. Esta técnica se usa en algoritmos conocidos como MergeSort, QuickSort y FFT. Además de simplificar la resolución de problemas complejos, permite paralelizar tareas y optimizar el rendimiento.

**Particionamiento**

El particionamiento consiste en dividir datos, bases de datos o sistemas en partes independientes llamadas particiones o shards. Esto permite distribuir la carga entre varios servidores y mejorar la escalabilidad. Por ejemplo, en bases de datos distribuidas, cada shard puede manejar un conjunto de datos específico, facilitando el manejo de grandes volúmenes y evitando cuellos de botella.

**Técnica de Pipelining**

El pipelining es una técnica en la que se organizan tareas en etapas secuenciales que se ejecutan de manera solapada. Mientras una etapa procesa un dato, la siguiente ya empieza con otro. Esto es muy usado en procesadores, sistemas distribuidos y pipelines de CI/CD, ya que permite un flujo continuo de trabajo y reduce el tiempo total de ejecución.

**Aceleramiento en cascada**

El aceleramiento en cascada consiste en encadenar varias optimizaciones o aceleradores, donde cada nivel mejora la eficiencia del siguiente. Por ejemplo, un sistema puede usar cachés, microservicios y procesamiento con GPU en serie, acelerando el flujo completo de trabajo. Esta técnica se aplica para aumentar rendimiento y reducir latencia en procesos complejos.

**Técnica Symmetry Breaking**

Symmetry Breaking se usa en algoritmos distribuidos para evitar que varios nodos realicen la misma acción al mismo tiempo, lo que podría causar conflictos. Romper la simetría permite que solo un nodo actúe como líder o encargado de una tarea. Es fundamental en la coordinación de sistemas distribuidos, asignación de recursos o elecciones de líderes.

**Cluster**

Un cluster es un conjunto de máquinas o nodos que trabajan juntos como si fueran un solo sistema. Se usan para aumentar la disponibilidad, distribuir la carga y procesar tareas en paralelo. Ejemplos típicos incluyen clústeres de Kubernetes, bases de datos distribuidas o clústeres de cómputo de alto rendimiento.

**Balanceo de cargas**

El balanceo de cargas distribuye el trabajo o las peticiones entre varios servidores para evitar que alguno se sobrecargue. Puede hacerse a nivel de red, aplicación o base de datos. Herramientas como NGINX, AWS ELB o HAProxy ayudan a mantener el sistema rápido, estable y escalable.

**Sistema Operativo**

El sistema operativo es el software que gestiona el hardware y los recursos del computador. Se encarga de la memoria, los procesos, los archivos y los dispositivos, funcionando como intermediario entre las aplicaciones y el hardware. Ejemplos son Linux, Windows y macOS.

**Procesador de tareas**

El procesador de tareas es el componente que ejecuta trabajos o procesos. Puede ser un CPU, un hilo de ejecución o un job scheduler como Celery o Airflow. Este se encarga de manejar tareas en paralelo, en segundo plano o distribuidas, asegurando que los procesos se ejecuten de manera ordenada y eficiente.

**Nexus**

Nexus es un repositorio de artefactos que centraliza la gestión de librerías, binarios e imágenes Docker. Permite mantener versiones controladas de dependencias y facilita su distribución y despliegue. Es muy usado en entornos empresariales para mantener consistencia en los proyectos.

**Docker**

Docker es una herramienta que permite empaquetar aplicaciones con todas sus dependencias en contenedores. Esto garantiza que el software funcione de la misma manera en desarrollo, pruebas y producción. Las imágenes de Docker se pueden almacenar en repositorios como Docker Hub o Nexus y se despliegan fácilmente en cualquier servidor.

**Repositorio de imágenes**

Los repositorios de imágenes son lugares donde se almacenan y versionan las imágenes de contenedores. Estos repositorios permiten compartir y desplegar contenedores de manera rápida y segura. Ejemplos son Docker Hub, GitHub Container Registry, AWS ECR o Nexus.

**SNS / SQS**

SNS y SQS son servicios de mensajería de AWS. SNS (Simple Notification Service) permite enviar notificaciones a múltiples suscriptores, mientras que SQS (Simple Queue Service) mantiene una cola de mensajes que se procesan de manera asincrónica. Se usan para desacoplar servicios y manejar eventos o tareas en segundo plano.

**Pub/Sub**

El modelo Pub/Sub (publicador-suscriptor) permite que un servicio publique mensajes y otros servicios se suscriban para recibirlos. Esto facilita la creación de sistemas distribuidos y reactivos donde los componentes no dependen directamente entre sí.

**GitHub Actions**

GitHub Actions es una plataforma de automatización integrada en GitHub. Permite crear workflows que ejecutan tareas como pruebas, compilación o despliegue cuando ocurre un evento, por ejemplo, un push o un pull request. Es muy útil para implementar CI/CD de manera rápida y sencilla.

**Jenkins**

Jenkins es un servidor de automatización muy usado para CI/CD. Permite construir pipelines complejos para integrar código, ejecutar pruebas y desplegar aplicaciones. A diferencia de GitHub Actions, Jenkins se autohospeda y ofrece más control sobre la infraestructura y los procesos internos.

**Ejemplos**

**Docker**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Pipeline de Jenkins**

A screenshot of a computer

AI-generated content may be incorrect.

pipeline {

    agent any

    stages {

        stage('Build image') {

            steps {

                bat 'docker build -t sns-sqs-api .'

            }

        }

        stage('Run container') {

            steps {

                bat '''docker run -d -p 3003:3003 -e AWS\_ACCESS\_KEY\_ID=secreto J -e AWS\_SECRET\_ACCESS\_KEY=secreto

 -e TOPIC\_ARN=arn:aws:sns:us-east-1:390809067334:pedidos-topic -e QUEUE\_URL=https://sqs.us-east-1.amazonaws.com/390809067334/pedidos-cola -e PORT=3003 --name sns-sqs-api sns-sqs-api'''

            }

        }

    }

}

**SNS y SQS**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Código de mi api que integra el uso de SNS y SQS**

require('dotenv').config();

const express = require('express');

const AWS = require('aws-sdk');

const app = express();

app.use(express.json());

const sns = new AWS.SNS({

    region: 'us-east-1',

    accessKeyId: process.env.AWS\_ACCESS\_KEY\_ID,

    secretAccessKey: process.env.AWS\_SECRET\_ACCESS\_KEY

});

const sqs = new AWS.SQS({

    region: 'us-east-1',

    accessKeyId: process.env.AWS\_ACCESS\_KEY\_ID,

    secretAccessKey: process.env.AWS\_SECRET\_ACCESS\_KEY

});

const topic = process.env.TOPIC\_ARN;

const queueUrl = process.env.QUEUE\_URL;

app.post('/enviar-pedido', async (req, res) => {

    const { mensaje } = req.body;

    if (!mensaje) {

        return res.status(400).json({ error: 'El campo "mensaje" es requerido' });

    }

    const params = {

        Message: mensaje,

        TopicArn: topic

    };

    try {

        const data = await sns.publish(params).promise();

        console.log('Mensaje enviado, ID:', data.MessageId);

        res.json({

            success: true,

            messageId: data.MessageId,

            mensaje: mensaje

        });

    } catch (err) {

        console.error('Error enviando mensaje:', err);

        res.status(500).json({ error: 'Error al enviar el mensaje' });

    }

});

app.get('/obtener-mensaje', async (req, res) => {

    const params = {

        QueueUrl: queueUrl,

        MaxNumberOfMessages: 1,

        WaitTimeSeconds: 5

    };

    try {

        const data = await sqs.receiveMessage(params).promise();

        if (data.Messages && data.Messages.length > 0) {

            const msg = data.Messages[0];

            const body = JSON.parse(msg.Body);

            const mensaje = body.Message;

            await sqs.deleteMessage({

                QueueUrl: queueUrl,

                ReceiptHandle: msg.ReceiptHandle

            }).promise();

            console.log('Mensaje procesado:', mensaje);

            res.json({

                success: true,

                mensaje: mensaje

            });

        } else {

            res.json({

                success: false,

                mensaje: 'No hay mensajes disponibles'

            });

        }

    } catch (err) {

        console.error('Error obteniendo mensaje:', err);

        res.status(500).json({ error: 'Error al obtener el mensaje' });

    }

});

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

    console.log(`API escuchando en puerto ${PORT}`);

});

**Prueba de API**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.