Simulación Basada en Agentes de Tráfico Urbano

María Angélica Alfaro Fandiño[1] and César Fernando Ortiz Rocha[2]

[1] *Escuela Colombiana De Ingenieria Julio Garavito,Bogotá,Colombia*
[1] *maria.alfaro@mail.escuelaing.edu.co*
[1] *cesar.ortiz@mail.escuelaing.edu.co*

Resumen— Este informe presenta la implementación y análisis de un sistema de simulación basada en agentes para el modelado de tráfico urbano, desarrollado en Java con capacidades de procesamiento paralelo. El sistema simula el comportamiento autónomo de vehículos en una red urbana con semáforos, permitiendo analizar patrones de congestión y flujo vehicular. Se implementaron dos versiones: una secuencial y otra paralela utilizando ExecutorService y ForkJoinPool. Los resultados experimentales demuestran que, debido a la naturaleza del problema (alta sincronización entre agentes y overhead de coordinación), la versión secuencial mantiene mejor rendimiento para cargas pequeñas y medianas (<200 vehículos). Sin embargo, la arquitectura paralela demuestra valor en simulaciones de gran escala (300 vehículos).

Palabras clave—Simulación basada en agentes, tráfico urbano, computación paralela, Java, HPC

Abstract— This article presentsthe implementation and analysis of an agent-based simulation system for urban traffic modeling, developed in Java with parallel processing capabilities. The system simulates the autonomous behavior of vehicles within an urban network with traffic lights, enabling the analysis of congestion patterns and traffic flow. Two versions were implemented: a sequential one and a parallel one using ExecutorService and ForkJoinPool. Experimental results show that, due to the nature of the problem (high synchronization among agents and coordination overhead), the sequential version maintains better performance for small and medium workloads (<200 vehicles). However, the parallel architecture demonstrates value in large-scale simulations (300 vehicles).

Keywords—Agent-based simulation, Urban traffic, Parallel computing, Java, High-Performance Computing

1. INTRODUCCIÓN

La simulación basada en agentes (ABM, Agent-Based Modeling) es una técnica computacional que modela sistemas complejos mediante la interacción de entidades autónomas llamadas agentes. En el contexto del tráfico urbano, cada vehículo es un agente que toma decisiones locales basadas en su entorno inmediato, emergiendo patrones globales de comportamiento del sistema. El crecimiento urbano y el aumento del parque automotor generan desafíos en la gestión del tráfico. Las simulaciones computacionales permiten:

Análisis predictivo:

Evaluar escenarios sin implementación física.

Optimización:

Probar configuraciones de semáforos y rutas.

Escalabilidad:

Simular ciudades completas con miles de agentes.

Este trabajo tiene como propósito el diseño e implementación de un sistema de simulación de tráfico urbano basado en agentes, capaz de modelar el comportamiento autónomo y reactivo de vehículos dentro de una red urbana estructurada en una malla de 50x50 celdas con calles, intersecciones y semáforos con ciclos periódicos. El simulador incorpora mecanismos de detección de colisiones, resolución de conflictos y métricas en tiempo real para evaluar flujo y niveles

de congestión. Con el fin de analizar tanto el comportamiento dinámico del sistema como su desempeño computacional, se desarrollaron dos versiones de la solución: una secuencial y otra paralela. Esto permitió comparar su eficiencia, evaluar los límites de escalabilidad y determinar el impacto del overhead asociado a la paralelización en distintos niveles de carga.

2. Estado del arte

En esta sección se abordan en detalle los conceptos fundamentales de la investigación, específicamente la simulación basada en agentes y la computación paralela.

A. Simulación Basada en Agentes

Un agente es una entidad autónoma capaz de interactuar de manera independiente con su entorno, percibiendo información a través de sensores, tomando decisiones basadas en reglas locales y actuando en consecuencia para modificar el estado del sistema en el que se encuentra. En el contexto de la simulación basada en agentes, cada agente opera de forma descentralizada, sin un control global explícito, y su comportamiento individual puede dar lugar a patrones emergentes a nivel macro. Este enfoque ha sido ampliamente estudiado en la literatura, donde se define a los agentes como sistemas computacionales situados en un entorno, capaces de acción autónoma para alcanzar objetivos o cumplir reglas definidas Michael Wooldridge [Wooldridge, 2009]. Asimismo, la modelación basada en agentes permite analizar cómo la interacción entre múltiples entidades simples puede generar dinámicas complejas, tal como se describe en los trabajos de Nigel Gilbert y Joshua M. Epstein sobre simulación social y sistemas complejos. [Gilbert, 2008, Epstein and Axtell, 1996]

B. Computación Paralela

La computación paralela tiene como objetivo acelerar la ejecución de un sistema mediante la distribución del trabajo entre múltiples núcleos de procesamiento que operan de manera concurrente. En términos generales, existen dos enfoques principales de paralelización: la paralelización por datos, que consiste en dividir el espacio de simulación en regiones o particiones que pueden procesarse simultáneamente, y la paralelización por tareas, que distribuye unidades de trabajo independientes —como agentes o procesos— entre diferentes hilos de ejecución. En el contexto de este proyecto, se adopta un enfoque de paralelización por tareas, donde cada hilo procesa un subconjunto de vehículos de manera concurrente, buscando mejorar el desempeño del simulador bajo cargas elevadas. Estos principios se fundamentan en la literatura clásica de computación paralela y arquitecturas multinúcleo, donde se analizan los modelos de descomposición del trabajo y sus implicaciones en escalabilidad y overhead de sincronización [Quinn, 2004, Rauber and Rünger, 2013].

C. Paralelización en ABM (Agent-Based Modeling)

La simulación basada en agentes (Agent-Based Modeling, ABM) es una metodología ampliamente utilizada para estudiar sistemas complejos en los cuales el comportamiento global emerge a partir de interacciones locales entre entidades autónomas.

La paralelización en modelos basados en agentes (ABM) presenta desafíos significativos debido a la naturaleza interactiva y compartida del entorno simulado. Uno de los principales problemas es la sincronización, ya que múltiples agentes interactúan sobre recursos comunes —como celdas o semáforos— lo que exige mecanismos de control para mantener la consistencia del estado global. Asociado a esto, pueden surgir condiciones de carrera cuando varios agentes intentan modificar simultáneamente el mismo recurso, comprometiendo la integridad de la simulación. Además, la paralelización introduce overhead computacional derivado de la creación y gestión de hilos, el uso de barreras de sincronización y el cambio de contexto entre procesos, lo cual puede reducir las ganancias de rendimiento esperadas. Finalmente, garantizar el determinismo y la reproducibilidad de los resultados constituye un reto adicional, ya que la ejecución concurrente puede generar variaciones en el orden de actualización de los agentes. Estos aspectos han sido ampliamente discutidos en la literatura de simulación paralela y sistemas concurrentes, donde se analizan los compromisos entre rendimiento, consistencia y escalabilidad [Fujimoto, 2000, Padua, 2011].

3. Arquitectura

El proyecto fue desarrollado en el lenguaje de programación Java, implementando tanto una versión secuencial como una versión paralela del sistema. La solución integra una simulación completa basada en agentes, donde los vehículos presentan comportamiento autónomo, capacidad de movimiento direccional inteligente y mecanismos de detección de colisiones y control de ocupación de celdas. Asimismo, se incorporó un sistema de semáforos ubicados en todas las intersecciones, con alternancia periódica entre los ejes Norte-Sur y Este-Oeste, y un ciclo de duración configurable. El entorno de simulación se construye a partir de una rejilla externa leída desde un archivo grid.txt, utilizando símbolos específicos para representar calles (.), intersecciones (+) y zonas bloqueadas (), lo que permite crear y modificar fácilmente distintas topologías urbanas. Finalmente, el sistema incluye un conjunto de métricas para el análisis del desempeño y comportamiento del modelo, tales como flujo promedio de vehículos, porcentaje de congestión, tiempo total de ejecución y throughput medido en movimientos por segundo.

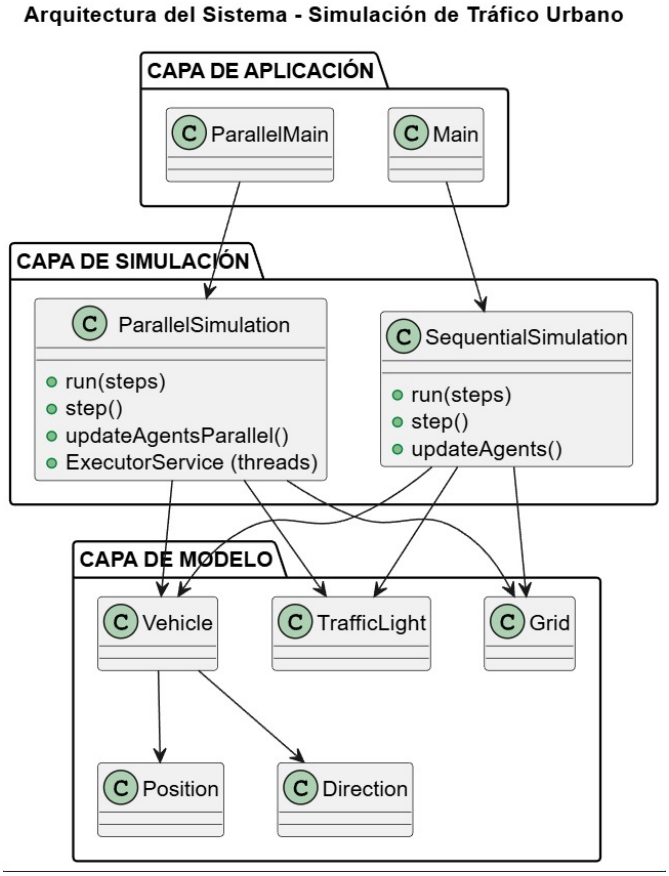

Fig. 1: Arquitectura de componentes

4. Clases Principales

A. Grid.java

Gestiona la representación espacial del entorno urbano

```
    // METODOS PRINCIPALES

    // Lee el archivo de configuracion
    public Grid(String filePath) throws
        IOException

    // Verifica si una posici n es
        transitable
    public boolean isTraversable(Position
        pos)

    // Verifica si una posici n es una
        intersecci n
    public boolean isIntersection(Position
        pos)

    // Obtiene todas las posiciones
        transitables
    public List<Position>
        getTraversablePositions()
```

B. Vehicle.java

Representa un agente vehículo con comportamiento autónomo.

```
    // METODOS PRINCIPALES

    // Mueve el veh culo a una nueva
        posici n
    public void move(Position newPosition)

    // Detiene el veh culo
    public void stop()

    // Cambia direcci n con probabilidad
    public void maybeChangeDirection(double
        probability)

    // Calcula siguiente posici n
    public Position getNextPosition()
```

C. TrafficLight.java

Controla el flujo de tráfico en intersecciones.

```
    // METODOS PRINCIPALES
    // Estado: Norte-Sur o Este-Oeste
    private Direction allowedDirection;

    // Contador interno
    private int cycleCounter;

    // Actualiza cada paso
    public void update() {
        cycleCounter++;
        if (cycleCounter >= cycleDuration) {
            cycleCounter = 0;
            toggleDirection();  // Alterna
                direcci n
        }
    }

    // Verifica si veh culo puede pasar
    public boolean canPass(Direction
        vehicleDirection) {
        return vehicleDirection ==
            allowedDirection ||
                vehicleDirection ==
                    allowedDirection.opposite
                    ();
    }
```

D. SequentialSimulation.java - ParallelSimulation.java

Motor principal de la simulación secuencial y paralela.

```
    // METODOS PRINCIPALES
    public void step() {

    // 1. Actualizar sem foros
    updateTrafficLights();

    // 2. Actualizar veh culos
    updateVehicles();

    // 3. Actualizar m tricas
    updateMetrics();
}
```

CONCLUSIONES

En cuanto al comportamiento del sistema, el modelo logra reproducir fenómenos característicos del tráfico urbano real, evidenciando dinámicas emergentes propias de sistemas complejos. En particular, se observa una densidad crítica de transición en el rango de mayor a 100 vehículos, a partir de la cual el flujo comienza a degradarse significativamente. En este punto emergen bloqueos en cascada y ondas de congestión que se propagan a través de la red, reflejando patrones similares a los observados en escenarios reales de tráfico vehicular.

En relación con el desempeño computacional, los resultados evidencian que la paralelización no resulta efectiva para cargas pequeñas y medianas, donde el costo adicional de coordinación supera los beneficios de la ejecución concurrente. El speedup máximo alcanzado fue de 0.55x con 300 vehículos utilizando 4 hilos, lo que indica que la versión paralela no logró superar el rendimiento de la versión secuencial en ese escenario. Este comportamiento se explica principalmente porque el overhead de sincronización domina el tiempo total de ejecución, reduciendo las ganancias esperadas por el uso de múltiples núcleos.

No todos los problemas se benefician automáticamente de la paralelización. En particular, los modelos basados en agentes con alta interdependencia entre entidades y granularidad fina en las operaciones requieren cargas de trabajo considerablemente grandes para compensar el overhead asociado a la sincronización y coordinación. En estos casos, la complejidad inherente de las interacciones puede limitar las ganancias de rendimiento, evidenciando que la paralelización debe evaluarse estratégicamente según la naturaleza del problema y no asumirse como una mejora universal.

REFERENCIAS

[Epstein and Axtell, 1996] Epstein, J. M. and Axtell, R. (1996). *Growing Artificial Societies: Social Science from the Bottom Up*. Brookings Institution Press, Washington, DC.

[Fujimoto, 2000] Fujimoto, R. M. (2000). *Parallel and Distributed Simulation Systems*. John Wiley & Sons, New York, NY.

[Gilbert, 2008] Gilbert, N. (2008). *Agent-Based Models*. Sage Publications, Los Angeles, CA.

[Padua, 2011] Padua, D., editor (2011). *Encyclopedia of Parallel Computing*. Springer, Boston, MA.

[Quinn, 2004] Quinn, M. J. (2004). *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, New York, NY.

[Rauber and Rünger, 2013] Rauber, T. and Rünger, G. (2013). *Parallel Programming: for Multicore and Cluster Systems*. Springer, Berlin, Germany, 2nd edition.

[Wooldridge, 2009] Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. John Wiley & Sons, Chichester, UK, 2nd edition.