INSTITUTO TECNOLÓGICO DE COSTA RICA
ESCUELA DE INGENIERÍA EN COMPUTACIÓN
PROGRAMA DE MAESTRÍA

Modelo para el manejo del control de la concurrencia en ambientes móviles considerando desconexiones frecuentes

Propuesta de Tesis sometida a consideración, para optar por el grado de Magíster Scientiae en Computación, con énfasis en Ciencias de la Computación

Juan de Dios Murillo Morera
Cartago, Costa Rica
Noviembre de 2014

Abstract

The problem of concurrency control in the context of mobile devices has become an important research topic, since in the current society both cell phones and tablets, and other devices have been used to perform transactions such as canceling a receipt phone, buy a movie ticket, consult a bank account.

The behavior of the crowd in mobile environments is a relatively new topic of study which has characteristics that differ from traditional approaches to database. The turnout in the computing environment appeared with the birth of operating systems with support for multiprogramming, and became a useful tool for getting the most out of a computer. Since then the competition has been applied to various computer applications such as databases, the Internet and multi-agent systems among others.

Parallel processing and management of large volumes of data are more common nowadays. Therefore, management of transactions to the device and between devices through the intervention of a web service is more common. In turn, there are several issues facing mobile environments transaction level. For example, frequent disconnections and low bandwidth in mind that sometimes hinder the proper handling of transactions and endanger both their integrity and concurrency.

To resolve this situation, have emerged algorithmic proposals which address the problem of concurrency control in mobile environments. However, they have always come limitations of any kind, because they use traditional methods for handling concurrency in mobile environments.

Among the proposals there are some that are characterized as conservati-

ve, or holding locks or latches that often are not required, or simply contribute to a process that leads to a slow transaction processing thus preventing many transactions arrive to commit.

Conservative proposals are used to manage mobile environments where the operations are local, thus achieving proper synchronization of these. For mobile devices, the problem begins in the time database transactions have not only local but also global, that is, there is a distributed heterogeneous environment. In situations like the one mentioned is where concurrency control becomes a difficult issue to resolve, especially when frequent disconnections occur.

The proposed research has the main objective to study the most important both pessimistic and optimistic algorithms, as well as some non-conservative representatives to propose a new approach based on the salient features of the studied streams that can resolve the problem of concurrency control in environments considering the frequent disconnects.

Resumen

El problema del control de la concurrencia dentro del contexto de dispositivos móviles se ha constituido en un importante tema de investigación, ya que en la sociedad actual tanto celulares y *tablets*, entre otros dispositivos, han sido utilizados para ejecutar transacciones tales como cancelar un recibo de teléfono, comprar una entrada al cine, consultar una cuenta bancaria.

El comportamiento de la concurrencia en ambientes móviles es un tema de estudio relativamente novedoso el cual presenta características que difieren de los enfoques tradicionales de bases de datos. La concurrencia en el ambiente computacional apareció con el nacimiento de los sistemas operativos con soporte para multiprogramación, y se convirtió en una herramienta útil para sacar el máximo provecho de un computador. Desde entonces la concurrencia ha sido aplicada a diversos usos informáticos como las bases de datos, la Internet y los sistemas multiagente entre otros.

El procesamiento paralelo y el manejo de grandes volúmenes de datos son más comunes en nuestros días. Por ende, el manejo de las transacciones en el dispositivo y entre dispositivos mediante la intervención de un *web service* es más común. A su vez, existen varios problemas que enfrentan los ambientes móviles a nivel transaccional. Por ejemplo, las desconexiones frecuentes y el bajo ancho de banda con el que se cuenta en ocasiones dificultan la correcta manipulación de las transacciones y ponen en peligro tanto su integridad como su concurrencia.

Con el fin de resolver esta situación, es que han surgido propuestas algorítmicas que garantizan resolver el problema del control de la concurrencia en ambientes móviles. Sin embargo, siempre han surgido limitaciones de

alguna índole, porque utilizan métodos tradicionales para el manejo de la concurrencia en ambientes móviles.

Dentro de las propuestas existen algunas que se caracterizan por ser conservadoras, o sea que llevan a cabo bloqueos o cerrojos que en muchas ocasiones no son requeridos, o bien simplemente contribuyen a un proceso que lleva a un lento procesamiento transaccional evitando así que muchas transacciones lleguen a *commit*.

Las propuestas conservadoras son utilizadas para administrar ambientes móviles donde las operaciones son locales, lográndose de esta manera una adecuada sincronización de estas. En el caso de los dispositivos móviles, el problema empieza en el momento en que las bases de datos no solamente tienen transacciones locales sino también globales, es decir, hay un ambiente distribuido y heterogéneo. En contextos como el mencionado es donde el control de la concurrencia se vuelve un asunto difícil de resolver, sobre todo cuando ocurren las desconexiones frecuentes.

La presente propuesta de investigación tiene como principal objetivo estudiar los más importantes algoritmos tanto pesimistas como optimistas, así como algunos representantes no conservadores, con el fin de plantear una nueva propuesta basada en las características más sobresalientes de las corrientes estudiadas que pueda resolver el problema del control de la concurrencia en ambientes considerando las desconexiones frecuentes.

www.tec.ac.cr

TEC Tecnológico de Costa Rica
Programa de Maestría
en Computación
Teléfono: 2550-2402

APROBACIÓN DE LA TESIS

"Modelo para el manejo del control de la concurrencia en ambientes móviles considerando desconexiones frecuentes"

TRIBUNAL EXAMINADOR

PhD. Carlos González Alvarado
Profesor Asesor

PhD. Jose Castro Mora
Profesor Lector

PhD. Mayela Coto Chotto
Profesor Externo

Dr. Roberto Cortés Morales
Coordinador del Programa
de Maestría en Computación

Noviembre, 2014

Dedicatoria

A mi madre Marta quien con su apoyo y enseñanzas ha hecho de mí un profesional consciente de su responsabilidad como docente universitario con nuestra Costa Rica.

A mi padre Manuel que aunque no esté conmigo en este mundo, sé que lo está espiritualmente. Agradezco su apoyo en todo el proceso de estudio. A él le dedico esta maestría.

A mi hijo Juan Camilo, quien es mi inspiración para realizar mi sueño de estudiar a profundidad el campo de las Ciencias de la Computación, en el Instituto Tecnológico de Costa Rica.

Agradecimientos

Al Dr. Carlos González Alvarado, mi tutor, por ser mi guía y más que eso por enseñarme el valor social del aporte de mi tesis como investigador y estudiante del Instituto Tecnológico de Costa Rica.

Al M.Sc. Julián González, filólogo de la Universidad de Costa Rica por su amable anuencia en la revisión filológica de la tesis.

Al Instituto Tecnológico de Costa Rica, por todo su apoyo en el proceso de finalización de mi tesis, a la Maestría en Computación y al Departamento de Orientación y Psicología, especialmente a la M.Sc. Laura Pizarro, por su apoyo en uno de los momentos más duros de mi vida: la muerte de mi padre Manuel en el año 2008.

Índice general

Índice de figuras

Índice de cuadros

Índice de Algoritmos

Capítulo 1

INTRODUCCIÓN

Los ambientes de computación móviles tienen acceso a la información mediante las conexiones inalámbricas. Según [5], la unidad móvil puede estar estacionaria, en movimiento o bien conectada intermitentemente a una red.

Conforme los avances tecnológicos se hacen presentes tanto a nivel de *software* como de *hardware*, es que se ha logrado que el acceso a la información contribuya a que actividades tales como pagar cuentas bancarias y transferir fondos se puedan dar desde cualquier sitio y cuando sea [6].

Actualmente, los dispositivos móviles se han constituido en un paradigma de relevancia dentro del desarrollo tecnológico, tal y como se ha estudiado en [7], en donde se establece lo siguiente: "Los avances en la comunicación inalámbrica y los dispositivos portables han creado un nuevo paradigma de computación donde los usuarios podrán acceder a la información desde diferentes ubicaciones físicas".

Dentro de este paradigma, el control de la concurrencia se ha constituido en un tema de gran interés dentro del campo de investigación de los MDBMS,[1] ya que los protocolos tradicionales para resolver los conflctos entre transacciones locales funcionan bien para bases de datos locales como pasa con *2PL*[2], pero no necesariamente para bases de datos distribuidas que son las utilizadas para dispositivos móviles, donde se incorporan transacciones globales que requieren de otro tipo de protocolos no tan conservadores.

[1] Sistema Administrador de Bases de Datos Móvil (del inglés Mobile Data Base Management System).

[2] Bloqueo de dos fases (del inglés two-phase-locking).

Según [3], en un ambiente de computación móvil, los usuarios pueden ejecutar transacciones en línea, independientes de su ubicación física. En un contexto móvil, diferentes *host* dispositivos pueden actualizar datos de forma simultánea, lo cual puede resultar en inconsistencia de la información. Aunque se han venido utilizando técnicas de bloqueos en este tipo de ambientes, no es la mejor solución por la condición de movimiento y de ubicación del dispositivo y de aspectos tales como el ancho de banda, las desconexiones frecuentes y las altas tasas de transacciones fallidas.

La investigación tiene como principal objetivo proponer un algoritmo que resuelva el problema del control de la concurrencia dentro de los ambientes móviles, tomando como base soluciones exitosas. Para ello, se ha optado por hacer un análisis comparativo de los diferentes algoritmos, de forma tal que se puedan estudiar tanto sus ventajas como sus desventajas y que el nuevo modelo sea híbrido y basado en las soluciones estudiadas.

Dentro de los algoritmos analizados, se pueden distinguir dos categorías: los conservadores o no agresivos y los no conservadores o agresivos. El primer grupo se caracteriza sobre todo por la forma en que llevan a cabo los bloqueos o cerrojos, ya que en muchas ocasiones son considerados innecesarios. A su vez, el segundo grupo se fundamenta en permitir que las operaciones se ejecuten pero manteniendo una fase de *precommit*, brindando una mayor libertad, sin perder de vista que pueda haber transacciones que deben evitarse por el conflicto que ocasionan.

El enfoque algorítmico que se propone en la investigación contempla las fortalezas de los algoritmos analizados, tanto conservadores como no conservadores, así como también algunas otras ideas que son producto de la investigación y experimentación haciendo novedoso el planteamiento de la propuesta al no utilizar cerrojos ni abortos y, además, considerar las desconexiones que pueda tener el dispositivo.

El algoritmo se fundamentará en un modelo formal, el cual tendrá un conjunto de definiciones y reglas que serán la base para la construcción algorítmica.

En cuanto al modelo formal, se explicará mediante una representación matemática el cumplimiento del control de la concurrencia, considerando las desconexiones que se puedan dar. Como un aporte adicional, el modelo matemático considerará la actuación humana dentro del proceso mediante la representación de un intervalo de tiempo, el cual es variable.

En el modelo computacional, se llevará a cabo la implementación de un simulador, que será el encargado de crear de forma virtual las situaciones en donde se den problemas de concurrencia y desconexiones frecuentes.

1.1. Definición del problema

El control de la concurrencia en bases de datos heterogéneas o para ambientes distribuidos, se ha constituido en un problema de investigación de gran interés dentro del campo de las ciencias de la computación [2], [5], [8],[9].

Según [10], "una transacción móvil es toda aquella transacción que se da desde un dispositivo móvil". Eso quiere decir que existen diferentes tipos de posibilidades de que se puedan dar inconvenientes tanto en *software* como en *hardware* al procesar las transacciones.

Como se menciona en [3], dentro de un entorno de dispositivos móviles existe una gran problemática por solucionar, ya que cada estación móvil puede actualizar en forma simultánea los datos, incurriendo así en la inconsistencia de estos.

Las transacciones móviles pueden gestarse bajo una serie de condiciones que producen la inconsistencia de los datos. Por otra parte, para [2], los entornos de *bases de datos distribuidas* que utilizan los ambientes móviles presentan una serie de incovenientes que son objeto de estudio: la variabilidad del ancho de banda, la reducción del espacio de almacenamiento, las desconexiones frecuentes, la fuente de alimentación baja de las baterías, etc.

Es por este motivo, como se ha mencionado previamente, que los autores citados dejan en claro que las desconexiones frecuentes y todo lo que implica su administración constituyen un problema que está siendo estudiado no solo por su impacto dentro de la administración de las transacciones, sino porque existe inconsistencia de datos muy evidentes, sobre todo cuando dichas desconexiones se dan de una manera constante, como suele pasar en nuestro país con la red de telefonía celular de empresas que ofrecen estos mismos servicios.

Los algoritmos conservadores resuelven el problema del control de la concurrencia, cuando se trata de los DBMS[3]. Sin embargo, estas propuestas no resuelven de una forma eficiente el control de la concurrencia cuando se trata

[3]Sistema Administrador de Bases de Datos (del inglés Data Base Management System).

del procesamiento de base de datos distribuidas en contextos móviles ya que emplean algoritmos conservadores que presentan bloqueos que no son muy adecuados para los ambientes distribuidos.

Por su parte, existen otras propuestas que sí resuelven el problema del control de la concurrencia contemplando las desconexiones frecuentes de una manera no conservadora, pero que en realidad por la tecnología que requieren es difícil su implementación en Costa Rica. Tal es el caso de ECHO[4][11] y BUC[5] [12]. Estas propuestas serán explicadas con detalle en el Capítulo 4.

Con base en la problemática planteada, es que nace la siguiente pregunta de investigación:

¿Qué características debe tener un modelo híbrido para el manejo del control de la concurrencia en ambientes móviles, tomando en cuenta las desconexiones frecuentes?

1.2. Justificación

La investigación se considera pertinente, ya que contribuiría a garantizar el control de la concurrencia en ambientes móviles tomando en consideración desconexiones frecuentes. Por desconexión se entiende la condición que se presenta cuando un equipo móvil es incapaz de comunicarse con algunos o todos los puntos de la red. A pesar del gran auge que han tenido las redes inalámbricas, aún siguen presentando una gran desventaja: son menos confiables que las redes cableadas.
De acuerdo con [13] se estima que la tasa de error de las redes cableadas presenta una magnitud de 10^{-10} por 10^{-4} de su contraparte inalámbrica, lo que propicia que de cada megabit transmitido, un kilobit será erróneo.
Es importante resaltar que en nuestro país las empresas que ofrecen servicios de telefonía celular presentan una red de telecomunicación con problemas en la señal a lo que se le añade lo irregular de nuestros suelos y formas de nuestros valles, llanuras, montañas, etc.

La razón más importante del planteamiento que se sugiere es que no hay al día de hoy una investigación hecha en este campo en nuestro país cuyas

[4]Método de Control de Concurrencia (del inglés Concurrency Control Method).
[5]Broadcasted and Updated Cycles.

características estén basadas en el no uso de cerrojos ni de abortos, constituyéndose así en una investigación novedosa para Costa Rica, siendo la más beneficiada toda aquella población que utiliza algún dispositivo móvil y que se ve afectada por las desconexiones frecuentes. Otra razón importante del porqué se puede considerar relevante la investigación, es que cada día que pasa más costarricences se suman a utilizar en sus diferentes actividades los dispositivos móviles, ya que les permite acceder a información de forma remota y concurrente, sin validar el correcto manejo transaccional que se debe tener para asegurar la concurrencia tanto de las transacciones que se están dando con el dispositivo como fuera de este.

La importancia del acceso cuando sea y donde sea según [8], se ha vuelto una realidad y una necesidad. Países como Japón, Alemania y China utilizan la tecnología móvil como parte de sus vidas cotidianas facilitando de esta manera la realización de actividades diarias, ya que no necesitan estar físicamente en los lugares donde deben efectuar las tareas sino que lo pueden hacer donde tengan conectividad.

Es por este motivo que nuestro país va cada día más hacia un desarrollo similar y varios son los proyectos que se desean culminar haciendo uso de la tecnología móvil. Así por ejemplo realizar el pago de servicios básicos como agua, luz, electricidad y telefonía, por medio de sistemas que estén ejecutándose en Internet dentro de las plataformas móviles son casos de su posible aplicación.

Sin embargo, nuestra red de telefonía móvil ha presentado una serie de deficiencias en cuanto a la señal brindada, a tal punto que según la ubicación donde se encuentre un ciudadano no existe señal o bien ocurren desconexiones frecuentes, de ahí que ante la necesidad de proporcionar un control de concurrencia de los datos es que sería muy útil desarrollar una investigación como la propuesta.

1.3. Objetivos

A continuación se describen los objetivos: general y específicos de la investigación.

1.3.1. General

Proponer un modelo para el manejo del control de la concurrencia en ambientes móviles considerando desconexiones frecuentes.

1.3.2. Específicos

- Identificar y comparar los principales algoritmos que resuelven el problema del control de la concurrencia en ambientes móviles.
- Crear un modelo formal que resuelva el problema del control de la concurrencia en ambientes móviles considerando desconexiones frecuentes.
- Crear un modelo algorítmico que resuelva el problema del control de la concurrencia en ambientes móviles considerando desconexiones frecuentes.
- Diseñar e implementar el modelo algorítmico planteado considerando desconexiones frecuentes.
- Evaluar el algoritmo que resuelve el problema del control de la concurrencia en ambientes móviles considerando desconexiones frecuentes.

1.4. Antecedentes

Seguidamente, se presentan algoritmos que resuelven tanto el problema del control de la concurrencia únicamente como los que lo hacen contemplando las desconexiones frecuentes.

1.4.1. El problema de la concurrencia en ambientes móviles

El control de la concurrencia es una de las más importantes maneras de administrar las transacciones que se ejecutan día con día en diferentes dispositivos en general y móviles en particular. La mayoría de los algoritmos que

resuelven el control de la concurrencia se basa en tres mecanismos: *locking* y *timestamps* los cuales consisten en establecer bloqueos sobre el esquema o tabla de la base de datos a utilizar (*locking*), así como estampillas de tiempo (*timestamps*) para garantizar un orden sobre las solicitudes que se llevan a cabo sobre cualquier esquema o tabla de la base de datos por parte de un dispositivo móvil. El problema radica en que esos mecanismos resuelven el control de la concurrencia, pero en historias locales; sin embargo, en ambientes distribuidos con historias globales no trabajan tan eficientemente ya que los dispositivos se ven afectados cuando por ejemplo se bloquea un esquema o tabla y se hace en forma general, sin considerar aquellas transacciones que no forman parte de un problema de concurrencia para un escenario específico.

Una primera propuesta es [3], en donde se trabaja el esquema *AVI*, el cual maneja como solución al problema la invalidación del caché. Este consiste en evitar el mecanismo de bloqueo tradicional que se ha empleado por error en los entornos móviles. Según [5], la movilidad y las desconexiones frecuentes han hecho que los métodos tradicionales no sean eficientes en este contexto.

A su vez, otras propuestas que garantizan el control de la concurrencia en ambientes móviles utilizando mecanismos tradicionales son los siguientes [8]:

Forced Conflicts Under Full Autonomy plantea que para alcanzar la serialización global es necesario el uso de etiquetas, consideradas como *ítems*. El objetivo del uso de etiquetas es poder establecer un orden a la hora de llevar a cabo las transacciones tanto de conflictos directos como de conflictos indirectos así como de transacciones locales y globales. La principal desventaja de este algoritmo es que en muchas ocasiones conduce a numerosos abortos, dado el uso de políticas conservadoras que provocan baja concurrencia.

Site Graph Algorithm plantea políticas conservadoras al igual que el caso anterior. Está basado en el uso de grafos no dirigidos para poder llevar a cabo transacciones concurrentes. Cuando ocurre algún conflicto, la transacción entra en un *delay* hasta que se termine el ciclo en el grafo. Al igual que el anterior, éste sufre de baja concurrencia, ya que un ciclo en el grafo no necesariamente indica un conflicto.

Locking Schemes plantea que la mayoría de los sistemas de bases de datos hace uso de este algoritmo para el control de la concurrencia a nivel de sitios locales. Está basado en la estrategia de bloqueos o cerrojos. El problema de esta estrategia radica en que cuando se producen bloqueos a nivel global,

estos pueden afectar también a nivel local y producir problemas en múltiples sitios, provocando así que se reduzca tanto la carga útil, a ambos niveles.

V-Lock representa un ejemplo de algoritmo de política no conservadora, el cual plantea el uso de tablas de bloqueo globales al tener un *wait-for-graph* global, utilizado para resolver potenciales bloqueos globales. Una de las desventajas de este algoritmo radica en que se ve afectado por el *overhead* de la red, es decir, por la baja tasa de transferencia. Por otra parte, está capacitado para poder regular la frecuencia de las señales en la comunicación. Trabaja con respuestas *ack* por cada una de las transacciones que han sido llevadas a cabo en forma satisfactoria y es considerado un grafo dirigido, en donde cada uno de sus nodos representa las transacciones. La misión del algoritmo es detectar ciclos haciendo uso del *wait-for-graph*. La política que utiliza es la de profundidad primero (DFS[6]).

Otras propuestas que resuelven el problema del control de la concurrencia se desarrollan en [14], [15], [16], [17].

1.4.2. El problema de la concurrencia en ambientes móviles enfocado en desconexiones frecuentes

Según [18], un ambiente de computación móvil usualmente se caracteriza por las desconexiones frecuentes. Las desconexiones frecuentes pueden aumentar la probabilidad de que las transacciones se pierdan o bien produzcan bloqueos innecesarios que bajan el redimiento transaccional de las operaciones globales en los diferentes sistemas de bases de datos.

Así por ejemplo, si una transacción está esperando un recurso que lo tenía otra transacción que se encuentra en estado de desconexión, un buen algoritmo debería permitir que el administrador de transacciones abortara las transacciones en estado de desconexión. Es importante recordar que estrategias tales como: *locking*, *optimism* y *time-stamp* tradicionalmente se han aplicado en ambientes distribuidos pero no han dado buenos resultados por ser estrategias conservadoras en los ambientes móviles, ya que se trata de entornos externos y distribuidos.

Por otro lado, el método ECHO propuesto en [11] es un ejemplo de protocolo para el control de la concurrencia en entornos distribuidos. Está basado en

[6]Búsqueda en Profundidad Primero (del inglés Depth-first Search)

la técnica *Information Broadcasting*, que se puede resumir en las siguientes características:

- Trabaja muy bien cuando la cantidad de sitios que serán accesados es muy grande.
- Cuanta mayor cantidad de información haya mejor es su desempeño.
- Está basada en mayor control de la concurrencia ante la réplica de los datos en diferentes bases de datos.
- Está compuesto, principalmente, por sitios *broadcaster*, sitios de usuarios finales y sitios intermedios.

El éxito del protocolo ha radicado en el uso del *Information Broadcasting*, conjuntamente con la idea de poder limitar el número de sitios que están en actualización, sin caer en bloqueos que pueden ser estrategias innecesarias y conservadoras. Esto no sería posible si solamente se utilizara la red de comunicación convencional. Otro de los aspectos importantes del método es que toma en cuenta la recepción de posibles fallos, o sea, que toma en cuenta las desconexiones frecuentes.

En este *framework* se encuentran los tres servidores de datos, los cuales son el sitio de *broadcaster*, el cual se encarga de enviar un mensaje *broadcast* a otros servidores. A su vez, el servidor de *broadcaster* para usuarios finales e intermedios facilita el uso de caché que utiliza el contenido *broadcast*.

Dentro de las limitaciones se encuentran las de los usuarios finales, los cuales tienen bastante limitada la capacidad cuando el contenido del *broadcast* necesita ser modificado o editado, para llevar a cabo réplicas sobre los sitios intermedios. Las réplicas mencionadas requerirán de control de concurrencia para así poder evitar múltiples versiones del mismo objeto *broadcaster*.

Dentro de los elementos que presenta la información están: multimedia con texto, vídeo o gráficos, sonidos o *hiperlinks* creados y editados por múltiples autores posiblemente en diferentes sitios. Debe estar disponible para prolongadas lecturas.

La idea básica del algoritmo es similar al *Optimistic*. El método permite actualizar transacciones que serán invocadas independientemente en cualquier sitio actualizador sobre cualquier réplica. Con el fin de evitar inconsistencias

en la actualización, la operación de actualización no llega hasta un estado de *commit*. Al igual que otros algoritmos, se hace uso del *timestamp* con el fin de validar si es posible o no poder llevar a cabo la actualización.

El método ECHO no se basa en algoritmos de control de la concurrencia fuertes, tales como el algoritmo de dos fases, ya que éstos son realmente eficientes en contextos en donde la cantidad de información no sea mucha, ya que la cantidad de bloqueos que se producirá será realmente impresionante según cambie la cantidad de sitios. Es por este motivo que el método ECHO hace uso de modelos de consistencia débiles, los cuales permiten un cierto nivel asincrónico y no requieren estar causando cerrojos o bloqueos por cada actualización que se ejecute sobre los sitios.

El *broadcasting* en ECHO es parte del intercambio de mensajes con el fin de coordinar las transacciones de actualización. A diferencia de los métodos tradicionales, los cuales únicamente están basados en comunicación con la red, este método si da soporte a sitios de actualización grandes. Está basado en *broadcasting*, técnica que permite que los mensajes puedan ser recibidos por más de un sitio al mismo tiempo.

Otra propuesta es BUC, el cual según [12], es un ejemplo exitoso donde se resuelve el problema del control de la concurrencia en ambientes móviles y, a diferencia de los modelos tradicionales, sí tiene mejor rendimiento en lo que respecta a desconexiones frecuentes.

La propuesta BUC se caracteriza por la técnica de *Broadcast and Updated Cycles*, la cual trabaja mediante ciclos de actualización de mensajes, algo similar a ECHO, haciendo uso de una técnica denominada *Data Broadcast*. Además, es importante señalar que resuelve los problemas de consistencia que se dan durante el proceso de *broadcast* y administra el *overlap* actualizando los *data items* en el servidor. Está basada en la arquitectura *data cycle*, la cual es propuesta para ambientes *broadcast*. Se fundamenta en que no es posible lograr la misma eficiencia en modelos tradicionales de control de concurrencia como lo es *2PL*, principalmente por los bloqueos que no vienen al caso en situaciones como desconexiones frecuentes y bajo ancho de banda.

Esta propuesta permite controlar mejor los abortos y evitar los bloqueos que hacen que se cree un gran *overhead*. Otros métodos empleados son el uso de *logs* de invalidación, el uso de grafos y los métodos de invalidación en general.

La principal característica de BUC consiste en dar un tiempo de respuesta eficiente en relación con la carga útil y reducir el *overhead* de la actualización de las transacciones. A diferencia de los métodos tradicionales, BUC si da un mejor soporte a problemas causados por la red, tales como las desconexiones frecuentes.

Una tercera propuesta es planteada por [18] en donde se expone el caso del protocolo *Check-Out/Check-In* que soporta ambientes con desconexiones frecuentes. En dicho protocolo se explica que los recursos de las estaciones móviles son limitados y que necesariamente hay que estar solicitando recursos al servidor. El problema se da básicamente cuando se está en periodo de desconexión, cuando el protocolo guarda en cola las transacciones que estaban en estado de desconexión y luego las actualiza en su base de datos sobre el servidor que está reparándose. Lo anterior lo hace basado en los datos de la cola *Check-In*.

Las propuestas anteriores comparten el hecho de resolver el problema del control de la concurrencia tomando o considerando las desconexiones frecuentes. El problema que presentan estas propuestas, es que están diseñadas o dirigidas a países con un alto nivel de desarrollo y con menos problemática en cuanto a la red de telefonía y telecomunicaciones como las que tenemos en Costa Rica. Situación que complica no solo su implantación en nuestro país sino también su adecuado uso. Es por este motivo, que personalizar una solución a la problemática planteada en esta investigación es lo más adecuado para garantizar el adecuado desempeño dentro de un contexto como el nuestro.

Con respecto a la organización del resto del documento, se hizo de la siguiente manera: el Capítulo 2 corresponde al Marco Teórico, donde se desarrollará la teoría necesaria para el entendimiento del problema; dentro de los apartados están: dispositivos móviles, control de la concurrencia en dispositivos móviles, MDBMS y ventajas y desventajas de propuestas algorítmicas que resuelven el problema del control de la concurrencia. Capítulo 3, Metodología, donde se explica la forma en que se alcanzarán los objetivos. Capítulo 4, se lleva a cabo un análisis comparativo entre los principales algoritmos que resuelven el problema del control de la concurrencia en ambientes móviles. Capítulo 5 corresponde al modelo Formal y Computacional; en esta sección se presentará la base matemática y algorítmica que sustenta el nuevo modelo. En el Capítulo 6 denominado Diseño Experimental, se describen los experimentos llevados a cabo así como las métricas utilizadas para la evaluación del desempeño del

modelo. Por otra parte, en el Capítulo 7 se estudia el Análisis de Resultados, que corresponde al apartado donde se discute sobre los resultados obtenidos respecto al modelo planteado y los otros modelos ya existentes. Finalmente, en el Capítulo 8 se presentan las conclusiones del trabajo, así como las líneas futuras para desarrollar tesis o trabajos de investigación sobre el tema.

Capítulo 2

MARCO TEORICO

Para desarrollar la propuesta de la presente investigación, fue necesaria la recopilación bibliográfica de algunos autores que han investigado en el campo de los dispositivos móviles, así como del control de la concurrencia. A su vez, se caracterizarán algunas de las propuestas de algoritmos que han resuelto el problema del control de la concurrencia dentro del contexto de las desconexiones frecuentes.

2.1. Dispositivos móviles

2.1.1. Definición

Una posible definición de los dispositivos móviles es: "...Los agentes móviles son aquellos que son capaces de trasmitirse ellos, tanto el programa como el estado de los atributos, a través de una red de computadoras"[4]. Estos dispositivos interactúan dentro de un ambiente móvil, el cual involucra el acceso de información a través de conexiones inalámbricas [1].

A su vez, el acceso de la información en esta clase de dispositivos debe darse: "...en cualquier momento y a donde sea"[1]. Según la frase anterior, el acceso a la información en los dispositivos móviles no tiene un lugar fijo, ni un momento determinado, lo cual implica mayor dificultad en el control de la concurrencia, ya que se presentan muchas interrupciones que pueden causar la inconsistencia de los datos. La forma de acceso a la información es muy similar a la de la tecnología de bases de datos distribuidas, donde el entorno distribuido juega un papel indispensable.

La idea del uso de las terminales móviles radicó básicamente en la sustitución de los RPC,[1], los cuales presentan una gran limitante y es que dados dos procesos como se aprecian en la Figura 2.1 sección (a), cuando se comunican un cliente y un servidor a través de la red, si el proceso cliente envía un mensaje al proceso servidor y este no lo contesta, entonces el proceso cliente quedará indefinidamente esperando a que el proceso servidor lo haga, lo cual podría resultar bastante caro en relación con tiempo y costo de procesamiento.

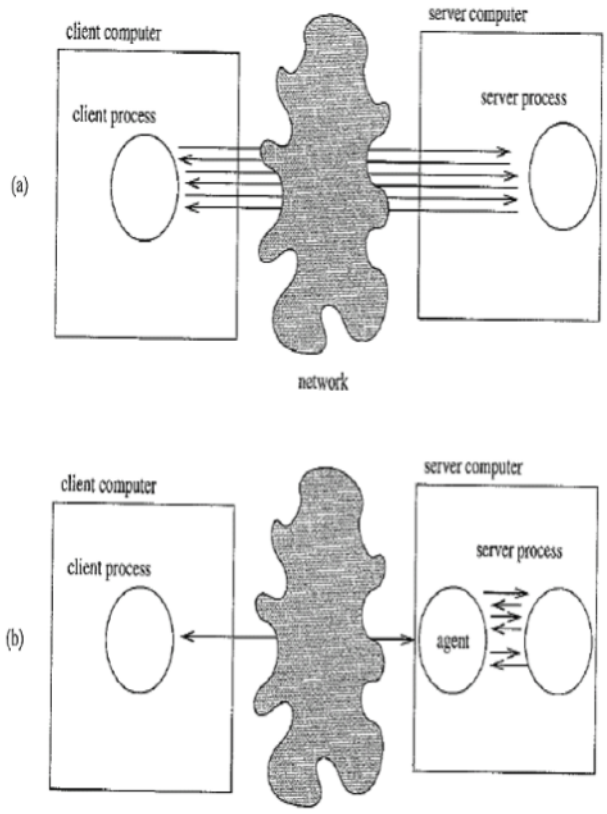


Figura 2.1: Modelo de RPC vs Modelo dispositivos móviles [1]

[1]Llamados a Procedimientos Remotos (del inglés Remote Procedure Call)

En la Figura 2.1 sección (a), se aprecia el clásico modelo cliente-servidor basado en *RPC*, en donde un proceso cliente trasmite información o datos a través de la red a la plataforma donde está el proceso servidor.

Por otra parte, en el caso de la Figura 2.1 sección (b), en vez de invocar un método, se envía un programa el cual interactuará con el proceso del lado del servidor, compartiendo el mismo espacio de dirección.

De esta manera, cuando el dispositivo ha terminado las interacciones, este retorna al cliente como se ilustra en la Figura 2.1 sección (b), con el resultado requerido, haciendo un uso mucho más eficiente de los recursos de la red que el del modelo clásico. Lo anterior favorecerá enormemente el buen desempeño del dispositivo, debido a que estos dispositivos tienen grandes problemas en lo que respecta al envío de la información por el ancho de banda y la latencia de la red, en un momento de mucho tráfico de paquetes.

Es así como el modelo que emplea agentes, Figura 2.1 sección (b), solamente ocupará la red cuando se requiera enviar el agente de la plataforma donde esté el recurso, y cuando haya terminado la o las tareas que se asignaron correspondería su devolución a la plataforma desde donde se envió originalmente.

El modelo de los agentes móviles es mucho mejor que el de *RPC*, ya que permite que dispositivos móviles tales como: *hand-held*, *palm* y *smartphones* no gasten tanto recurso al hacer los llamados en forma remota, sino que el agente esté ya sea del lado del cliente o del servidor según se requiera [4].

En el caso de los agentes móviles, se trasmite tanto el agente como el estado del programa conjuntamente con el contador de programa [4]. Este último se utiliza para que el agente sepa dónde quedó la última vez.

2.1.2. Características de los dispositivos móviles

A continuación se presenta un resumen de las principales características de un dispositivo móvil.

Cuadro 2.1: Características de los dispositivos móviles [4]

Característica	**Descripción**
Interoperabilidad heterogénea	*Software* y *hardware* heterogéneos
Transacciones y control de concurrencia	Se basa en el principio ACID
Distribución transparente	Debe ser portable e interoperable
Representación transparente	Diferentes nombres y formatos
Búsqueda Inteligente y navegación de Datos	Interfaces y búsquedas eficientes
Localización transparente	Datos transparentes para usuarios
Dependencia de la localización	Datos dependientes del usuario o sistema
Soporte a conexiones débiles y desconexiones	Constante acceso a los datos
Serialización	Estados de las variables del móvil
Ejecución local y remota	Lugar y características del agente
Seguridad	Vulnerabilidad, latencia y procesador
Cambios constantes	Constantes cambios remotos

Dentro de las características fundamentales de los dispositivos están: interoperabilidad, localización, búsqueda inteligente y navegación, serialización, ejecución local y remota. Dichas características son de gran importancia en contextos en los que se necesita una comunicación remota con el fin de llevar a cabo tareas transaccionales.

2.1.3. Arquitectura de las redes de dispositivos móviles

Según [2], una red de dispositivos móviles es vista como un conjunto de elementos que interactúan entre si. Dentro de estos elementos, se tienen como principales: *Mobile Host* y *Fixed Host*. El *Mobile Host*, corresponde a la estación móvil de trabajo, mientras que el *Fixed Host* corresponde al servidor o web service encargado de procesar las solicitudes hechas por las estaciones de trabajo.
A continuación se presentan un esquema que resume la iteración entre el *Mobile Host* y *Fixed Host*.

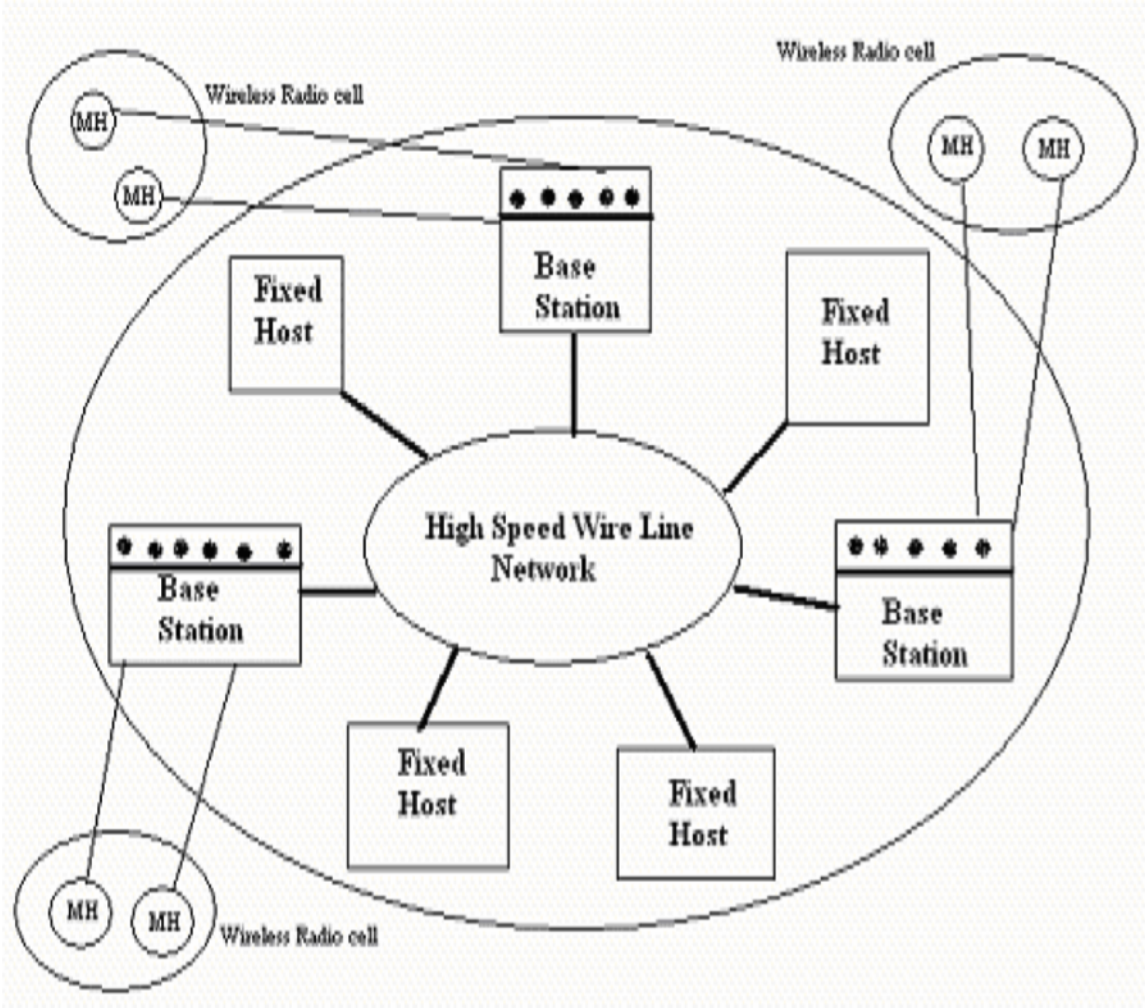


Figura 2.2: Arquitectura de un red móvil [2]

2.1.4. Tipos de dispositivos móviles

Básicamente se manejan dos tipos: los estacionarios y los móviles. Para efectos de la investigación, se explicarán en detalle los móviles o en movimiento [1]. Los dispositivos móviles deben ser capaces de adaptarse a las condiciones de cambio [1]. Dentro de esas condiciones de cambio se pueden citar las siguientes:

- El ambiente de red.
- Los recursos disponibles de la aplicación.

Es importante destacar que la autonomía del dispositivo y la dependencia hacia el servidor del cual obtiene la información es crítica, ya que pueden existir factores que ocasionen problemas en el manejo del control de la concurrencia, como lo son:

- Las frecuentes desconexiones de la red.
- El bajo ancho de banda.
- Las restricciones de poder.

Dado que el ambiente es cambiante en los dispositivos móviles, las aplicaciones deben adaptarse al nivel de soporte que puedan brindar sistemas estacionarios como los servidores. El concepto de móvil implica mucho más diversidad de rangos y cantidades de acceso a datos [1]. Por lo tanto, los ambientes remotos deben acceder información que se encuentra localizada en diferentes gestores de base de datos, lo cual dificulta no solo su procesamiento sino también su control.

Al igual que los sistemas móviles, los sistemas administradores de base de datos distribuidas presentan una capa para un sistema global la cual permite el acceso distribuido a múltiples y preexistentes sistemas de bases de datos locales. De ahí la importancia de tener que serializar el control de la concurrencia en un ambiente como el descrito anteriormente, ya que aunque haya un buen manejo del control de la concurrencia en relación con los sistemas de bases de datos locales, si no se serializa nada garantiza que se mantenga la concurrencia de los datos, ya que la historia local estaría desfasada de la historia global.

Los MDBMS[2] son los que permiten el acceso a largas cantidades de datos sobre un medio inalámbrico. El acceso a los datos por parte de múltiples usuarios ocasiona que haya que administrar concurrentemente las transacciones. La importancia de los MDBMS es que permiten lograr una coordinación adecuada entre el sistema global y cada uno de los sistemas administradores locales, permitiendo así mantener el control de la concurrencia en forma adecuada y que tanto el *software* como el *hardware* de diferentes sistemas locales sean transparentes para el usuario. Cada uno de los sistemas locales presenta un esquema de integración [1], el cual permite una vista lógica y global de la información.

2.1.5. Área de aplicación

Dentro de las áreas de aplicación de los dispositivos móviles se tienen las siguientes: telefonía celular para comunicación, transportes de trenes y aeroplanos que comunican datos locales con sistemas supervisores, así como el acceso a información Web ya sea para uso personal o de negocios, entre otros [19].

2.2. El control de la concurrencia

2.2.1. Definición

El control de la concurrencia se puede definir como el proceso que:
"...coordina las operaciones de múltiples transacciones que operan en paralelo y acceden datos compartidos" [1].
"El control de la concurrencia se usa como significado de incremento de la carga útil y reduce el tiempo de respuesta" [1].

En el caso de los dispositivos móviles, el acceso compartido entre datos se vuelve una tarea difícil de controlar, ya que se tienen diferentes sistemas administradores de bases de datos a nivel local con datos que pueden variar de un sistema local a otro.

[2]Sistema Administrador de Bases de Datos Móvil (del inglés Mobile Data Base Management System)

2.2.2. El modelo transaccional móvil

El modelo transaccional móvil se caracteriza por tener transacciones tanto locales como globales. Las transacciones locales son las que pertenecen a los sistemas administradores de bases de datos locales, mientras que las globales son aquellas que no están específicamente en un sistema administrador, sino que pertenecen a todos.

En la Figura 2.3 se representa el modelo local, el cual para lograr el control de la concurrencia utiliza una serie de colas (CPU, IO, Comunicación) que permiten la administración de los recursos por parte del Sistema Operativo. La salida de las colas anteriores conduce a mensajes que son enviados a un sistema global para el manejo de transacciones globales.

Es importante rescatar que los modelos locales reciben transacciones del modelo global como se aprecia en la Figura 2.4, y estas son conducidas a una cola de transacciones listas y activas.

Finalmente, otro de los aspectos de importancia de este sistema radica en el manejo de los bloqueos mediante el uso de colas de inicio y de bloqueo respectivamente, garantizando de esta manera administrar las transacciones que están en estado de bloqueo.

A su vez, la Figura 2.4 permite interpretar que cada sistema administrador de bases de datos local tiene su propia administración del control de la concurrencia. Ahora bien, tanto el sistema global como el local se intercambian información como se ha mencionado, siendo esta información (transacciones) el objeto de estudio de la presente investigación, específicamente cuando ocurren las desconexiones frecuentes. En el modelo global, las transacciones de los MDBMS locales se constituyen en las entradas mediante mensajes en el modelo global. Para una mejor administración del modelo se tiene un planificador, el cual administra las operaciones: *write, read, commit*y *abort*. El planificador, recibe transacciones tanto de la cola de activos como de la de listos.

El esquema o modelo global opera de manera similar que el local, siendo de mucho más difícil administración el control de la concurrencia ya que se deben sincronizar las transacciones locales de cada MDBMS.

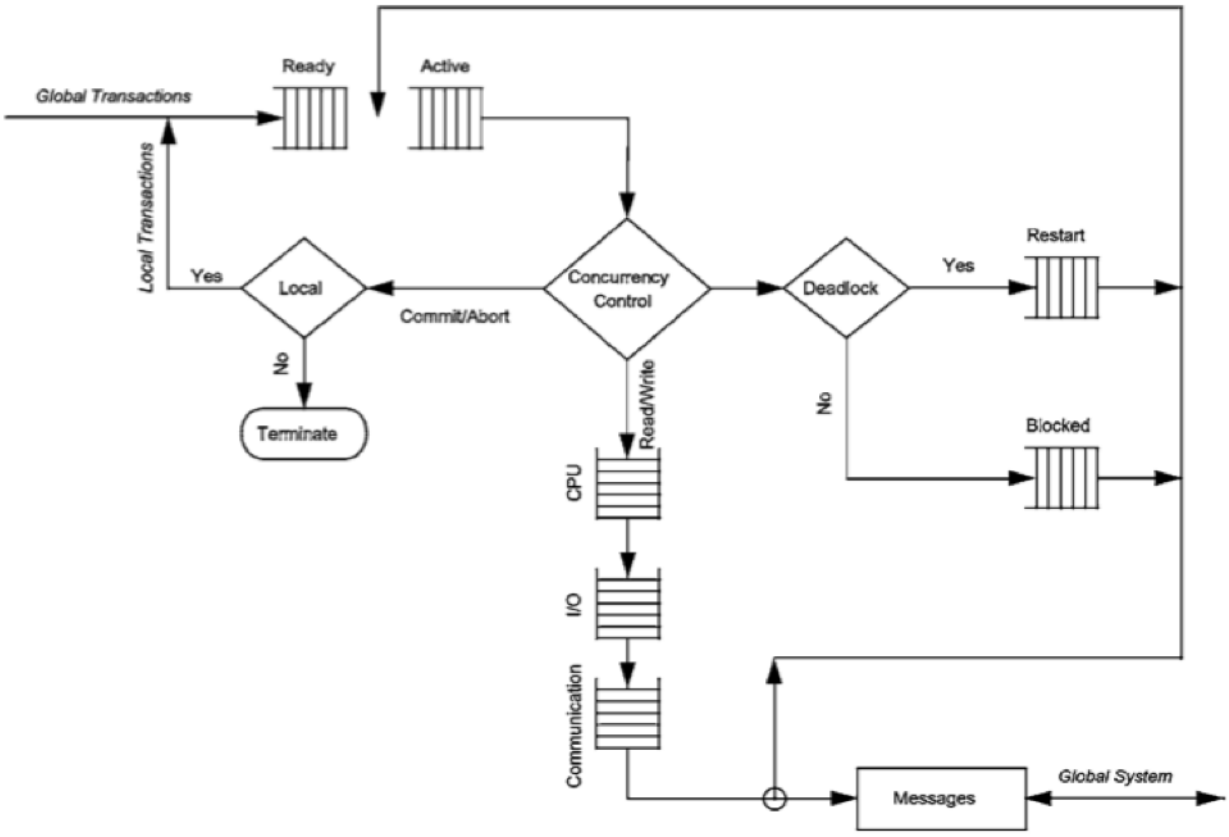


Figura 2.3: Esquema de transacciones locales [1]

Es importante destacar el hecho de que los manejos locales y globales son diferentes, que no es lo mismo el manejo del control de la concurrencia en las transacciones globales que las locales, ya que la historia global es más difícil de serializar que las locales y estas últimas poseen un sistema administrador propio.

Existen cinco tipos de transacciones en el ambiente de las bases de datos móviles según [19]:

- Las transacciones móviles.
- Las subtransacciones.

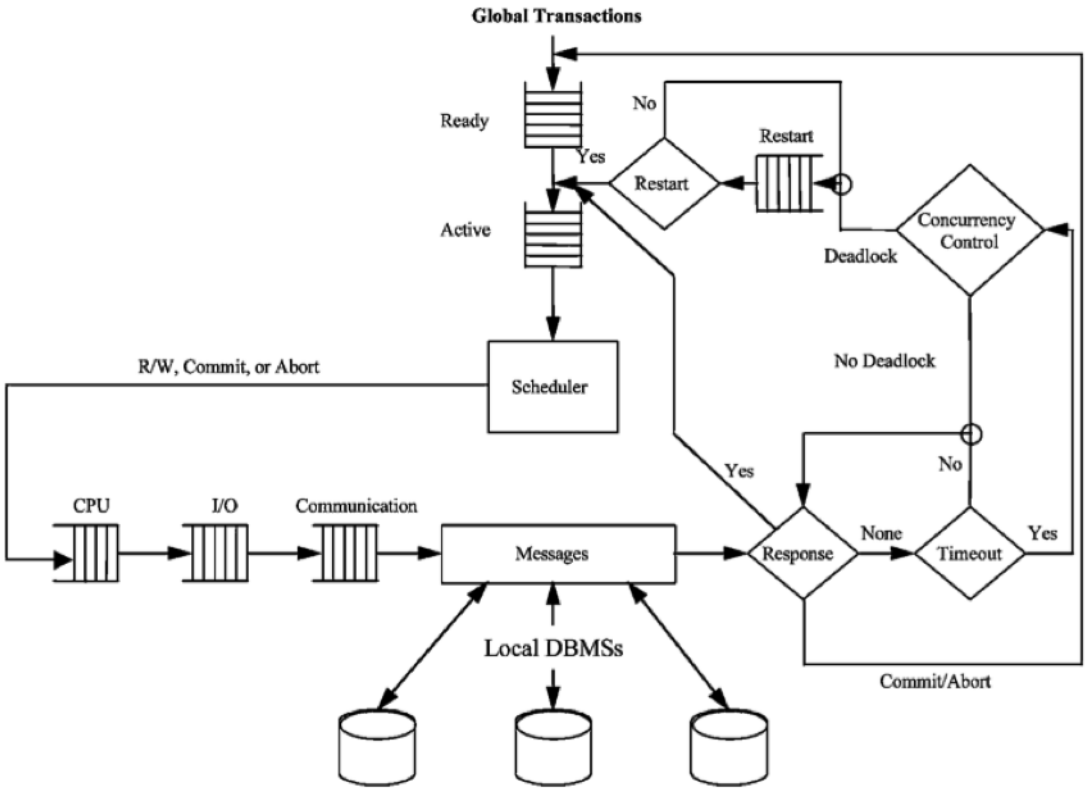


Figura 2.4: Esquema de transacciones globales [1]

- Las transacciones globales.
- Las subtransacciones globales.
- Las transacciones locales.

Cada vez que un dispositivo móvil envía una transacción a la base de datos, ésta es dividida en otras subtransacciones, las cuales pueden ser transacciones locales dentro de una historia global. Por otra parte, tenemos los *MDAS*[3], encargados de proveer acceso global sin requerir un conocimiento preciso del acceso local; así el sistema podrá procesar una consulta de usuario en sus propios términos.

[3]Sistema de acceso móvil de datos (del inglés Mobile Data Access System)

Según [19], cada subtransacción móvil incluye una o más transacciones globales. Cada una de las transacciones globales representa transacciones locales con sus propios recursos. Para un MDBMS local, las subtransacciones globales son tratadas como cualquier otra transacción local [19].

Las transacciones son manejadas en una estructura de datos conocida como árbol, en donde la transacción de la raíz es la que se conoce como la transacción de más alto nivel por ser la que está en el nivel más arriba, es decir, en el nivel cero. Las demás transacciones son conocidas como subtransacciones.

Al igual que los árboles convencionales, los niveles más bajos corresponden a las hojas, que son aquellos nodos que no tienen subtransacciones. El hijo del mismo padre se conoce como *siblings*.

2.2.3. La administración transaccional y el control de la concurrencia

El modelo del control de la concurrencia debe satisfacer las propiedades *ACID*[4], que corresponden a: Atomicidad, Consistencia, Aislamiento y Durabilidad [1]. Uno de los principales problemas de este modelo en el contexto de los ambientes móviles es que se hace difícil mantener la autonomía ya que las operaciones locales están fuera de control del sistema global. La desventaja es que se pueden producir grandes desfases, frecuentes abortos, inconsistencias y cerrojos según del protocolo que se seleccione. Otro de los aspectos por contemplar es que el esquema de administración de transacciones de un MDBMS debería garantizar correctitud bajo todas las circunstancias [1], es decir que la integridad de los datos se mantenga y esto se logra mediante el control de la concurrencia.

Un protocolo efectivo de control de la concurrencia debería entonces poder procesar correctamente y simultáneamente aquellas transacciones múltiples [1].

El control de la concurrencia en sistemas de bases de datos heterogéneos se refiere a la serialización de transacciones tanto globales como locales. Los sistemas globales tienen información sobre las transacciones globales, pero las transacciones locales no son vistas a nivel global [1]. Lo anterior ocasiona

[4]Atomicidad, consistencia, aislamiento y durabilidad(del inglés, atomicity, consistency, isolation, durability)

un problema de concurrencia entre la historia local y la historia global; de ahí la importancia de la serialización que contribuye a evitar este tipo de conflicto.

En cuanto al aislamiento, la forma más simple de implementarlo es correr solamente un programa al mismo tiempo como se menciona en [20]. Esto significa que todos los programas sean cortos, que todos los datos estén centralizados en la memoria principal y que todos los datos sean accedidos por un simple procesador.

Las demás propiedades son: atomicidad, que se refiere a que deben ser operaciones irreductibles; durabilidad, que se refiere a que durante el proceso de la transacción debe tener un período de tiempo establecido para que se lleve a cabo la transacción y, finalmente, debe ser consistente, lo cual significa que los resultados deben prevalecer en el tiempo y no darse inconsistencia de estos.

2.2.4. El control de la concurrencia en MDAS

La principal diferencia entre los MDAS y los MDBMS es la conexión en los servidores y los clientes a través de un ambiente inalámbrico y dispositivos que requieren del acceso a los datos [1]. Sin embargo, ambos entornos proveen el acceso a datos compartidos globales y en ocasiones as soluciones de MDAS sirven o se adaptan bien para MDBMS con tan solo contemplar movilidad y acceso inalámbrico.

Una transacción consiste en una colección de operaciones: *read(r), write(w), commit(c)* y *abort(a)* [1].

Los MDAS están conformados por una estructura jerárquica similar a la de los MDBMS. Una transacción global puede ser enviada por cualquier nodo de la jerarquía, ya sea por un nodo local o por un nodo esquema. Una vez que la transacción se recibe, esta se dividide en subtransacciones, las cuales se procesan en una estructura.

Es importante resaltar que la resolución de la coordinación está dada por un nodo coordinador que es el que contiene la información y es el encargado de manipular las transacciones globales [1]. A su vez, en los sistemas de bases de datos distribuidas, existe una tabla que mantiene un cerrojo de las

transacciones globales hasta que no se haya garantizado que se han serializado en un modelo bastante rígido, el cual ocasiona grandes esperas de otras transacciones que realmente no tienen por qué estar bloquedas.

Los MDAS coordinan las transacciones globales sin ninguna información de control de ningún MDBMS local [1].

2.2.5. El control de la concurrencia en MDBMS

Los sistemas administradores de bases de datos locales son autónomos; sin embargo, la información sobre la serialización no está disponible a nivel global. Los MDBMS deberán tener una historia serializable para un buen desempeño y ejecución de las transacciones concurrentes, lo cual implica que el orden de las historias locales debe ser consistente con respecto al de la historia global. Los MDBMS requieren atender conflictos directos e indirectos entre transacciones globales. En los MDBMS, cada una de las transacciones móviles, subtransacción, es etiquetada de acuerdo con el árbol donde se encuentran las transacciones. La consecuencia de violar el orden de las etiquetas tanto local como global es un aborto de las transacciones mismas. A continuación se ofrece un gráfico como muestra de una transacción móvil.

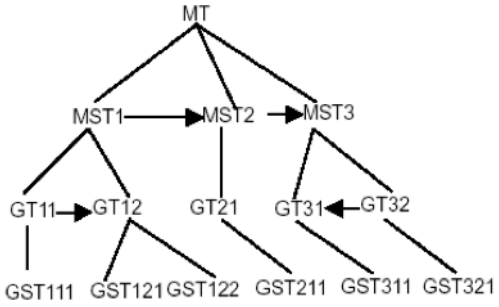


Figura 2.5: Ejemplo de una transacción móvil [1]

Según la figura 2.5, una transacción móvil (MT) está compuesta por varias subtransacciones móviles (MST). Cada MST puede operar en una o varias bases de datos locales. A su vez, cada MST está conformada por una o varias transacciones globales (GT) y éstas a su vez de una o varias subtransacciones globales(GST), ejecutadas en un sitio local.

2.2.6. Leyes del control de la concurrencia

Por otra parte, según [20], la consistencia es la propiedad estática, el control de la concurrencia es el problema, la serialización es la teoría y el cerrojo es la técnica.

Cada transacción debe verse como un estado consistente, pero puede haber ocasiones en que se ejecutan concurrentemente y las conductas de algunas pasan a ser inconsistentes [20].

La primera y la segunda ley de la concurrencia respectivamente son:

- La ejecución concurrente debería no causar malos funcionamientos en aplicaciones.
- La ejecución concurrente debería no tener bajo rendimiento o mucho más altos tiempos de respuesta que la ejecución serial.

La mayor parte de los malos funcionamientos se debe a que existen muchos programas con variedad de respuestas distintas entre muchos procesadores con muchos almacenamientos de memoria.

El control de la concurrencia es un mecanismo que presenta un alto *overhead*, lo cual puede ser bastante costoso [20]. La segunda regla favorece a algoritmos simples no complejos.

2.2.7. Las tres dependencias negativas

Los ciclos pueden tomar tres formas genéricas, a saber [20]:

- *Actualización perdida*

  Un ejemplo de este caso es: Supóngase que dos personas están escribiendo un programa juntas, cada una toma una copia del programa y lo cambia independientemente. Posteriormente se retorna el programa a la biblioteca de programas; el programa finalizará solamente con los cambios de la primera persona o solo con los cambios de la segunda, pero no con ambos, perdiéndose así una de las dos actualizaciones.

- *Lectura sucia*

  Continuando con el ejemplo anterior, supóngase que la primera persona instala tentativamente una versión del programa en la biblioteca y la otra la usa. Posteriormente, la primera persona se da cuenta de que tuvo un error y de que debe corregirlo reversando el cambio y creado así una nueva versión del programa. La lectura antes de llevarse a cabo el cambio sería lo que corresponde a una lectura sucia.

- *Lectura irrepetible*

  Con base nuevamente en el ejemplo anterior, supóngase que la primera persona hace una lectura de una versión 1, mientras más tarde se instala la versión 2 por parte de la segunda persona. Cuando la primera persona desee volver a hacer una lectura, encontrará que la versión ha variado, lo cual no le garantizará la consistencia de la lectura. Por ese motivo se define como irrepetible ya que esa lectura fue única en su momento.

Como se han explicado mediante los ejemplos dados en el apartado anterior, existen diferentes escenarios de la vida real, en la que se puede dar un problema de control de concurrencia sobre los datos que están siendo actualizados. De ahí, que el modelo propuesto, no solo deberá contemplar las actualizaciones perdidas, las lecturas sucias y las lecturas irrepetibles sino cualquier otra situación que ponga en peligro la concurrencia de los datos y de esta manera garantice la integridad de los mismos.

A continuación se presenta la metodología de la investigación donde se le explicará al lector el cómo se pretende cumplir con los objetivos planteados

Capítulo 3

METODOLOGÍA

En este apartado, se describen las diferentes fases que se siguieron para lograr el alcance de cada uno de los objetivos planteados, con el fin de proponer el modelo para el manejo del control de la concurrencia en ambientes móviles, tomando en cuenta las desconexiones frecuentes.

3.1. Primera fase

La primera fase consistió en investigar los principales algoritmos que cumplían con el manejo del control de la concurrencia en ambientes móviles. Esta fase está ligada al cumplimiento del primer objetivo secundario. La motivación de haber hecho este análisis previo de los algoritmos más representativos en el área ha sido poder contar con un cuadro-resumen o sumario el cual se ha desarrollado en el Capítulo 4 de la investigación, donde se plantearon las fortalezas y debilidades de cada uno de los doce algoritmos analizados.

Mediante el análisis de la caracterización de los algoritmos, lo cuales fueron divididos en tres categorías: pesimistas, optimistas y conservadores, se tomaron como insumo las fortalezas obtenidas y se tomaron muy en cuenta las debilidades para no plasmarlas en el nuevo modelo.

Como parte del cumplimiento de esta fase se planteó como indicador el estudio de un mínimo de cinco modelos. Sin embargo, es importante resaltar que se lograron analizar doce modelos en el contexto del problema del control de la concurrencia en dispositivos móviles. Llevándose a cabo siete estudios adicionales de los planteados originalmente.

3.2. Segunda fase

La segunda fase toma como base la primera con el fin de plantear un modelo tanto a nivel formal como algorítmico. Esta fase está ligada al segundo objetivo secundario y le dio el formalismo matemático a la propuesta del modelo. En el ámbito formal, se planteó el modelo mediante definiciones matemáticas relacionadas con las transacciones globales y locales propias de un modelo transaccional móvil. Como secciones, el modelo matemático contó con: sistema móvil, transacciones móviles, transacciones locales y globales, conflictos directos e indirectos, *fixed host*, *mobile host*, desconexiones frecuentes y latencia de la red.

Además, se planteó el algoritmo que representa la implementación que se hizo del modelo. La representación algorítmica representa la solución computacional del problema planteado. En este caso, se diseñó tanto una solución algorítmica para el cliente como para el servidor, con la diferencia que el servidor es el que contiene el monitor, el cual valida si existe o no problema transaccional a la hora de revisar las historias globales correspondientes a todas las transacciones locales de cada uno de los dispositivos.

3.3. Tercera fase

Se analizó e implementó un simulador tomando la implementación algorítmica y matemática planteada en la segunda fase. Esta fase está ligada al tercer objetivo secundario de la investigación y tiene como principal resultado el simulador que resuelve el problema planteado.

Para lograr el objetivo planteado en esta fase, fue necesaria la implementación en el lenguaje de programación Java, específicamente en el ambiente *Android* de los algoritmos planteados en la segunda fase. En relación con la implementación, se empleó el modelo cliente-servidor hilado y la virtualización de simuladores de celulares. Ver Anexo.

3.4. Cuarta fase

Con el fin de evaluar el desempeño del simulador, fue necesario el plantemiento de un diseño experimental, donde se definieron métricas de evaluación de desempeño (ver Capítulo 6) así como un modelo estadístico para analizar los

resultados (ver Capítulo 6). Esta fase está relacionada con el cuarto objetivo secundario cuyo producto esperado es la evaluación del desempeño del simulador desarrollado. Con el objetivo de cumplir con la evaluación propuesta se llevó a cabo un *benchmarking* (ver Capítulo 6) entre el algoritmo propuesto y uno de los algoritmos del estudio planteado (ver Capítulo 4). También se llevó a cabo un análisis únicamente sobre el algoritmo propuesto que incluía: tiempo promedio de commit, tasa de transacciones en conflicto y tasa de transacciones sin conflicto. La evaluación de estos dos tipos de evaluaciones de desempeño se encuentran detalladas en el Capítulo 7.

Es importante resaltar que esta metodología fue iterativa, por lo que las fases 1, 2, 3 y 4 se ejecutaron de forma secuencial hasta lograr el cumplimiento del objetivo general, es decir buscando lograr la eficacia del modelo. A su vez, los pasos 5,6, 7 y 8 se utilizaron con el fin de optimizar el resultado obtenido, es decir alcanzar eficiencia en el resultado del modelo.

Como se puede apreciar en la Figura 3.1, tanto para los pasos de construcción como de optimización del modelo siempre, se siguió la estrategia de seguir al paso anterior en el caso de la optimización y de avanzar al paso siguiente en el caso de la búsqueda de la solución; Lo cual permitió ir del análisis preliminar de los estudios hasta la evaluación de los resultados obtenidos relacionados con la comparación computacional y la evaluación de desempeño de la solución obtenida.

A continuación se presenta el análisis comparativo entre los estudios que resuelven el problema del control de la concurrencia. En este análisis se le explicará al lector las ventajas y desventajas de los diferentes modelos estudiados, los cuales han sido lo que han servido como base para el planteamiento de la nueva propuesta formal y algorítmica.

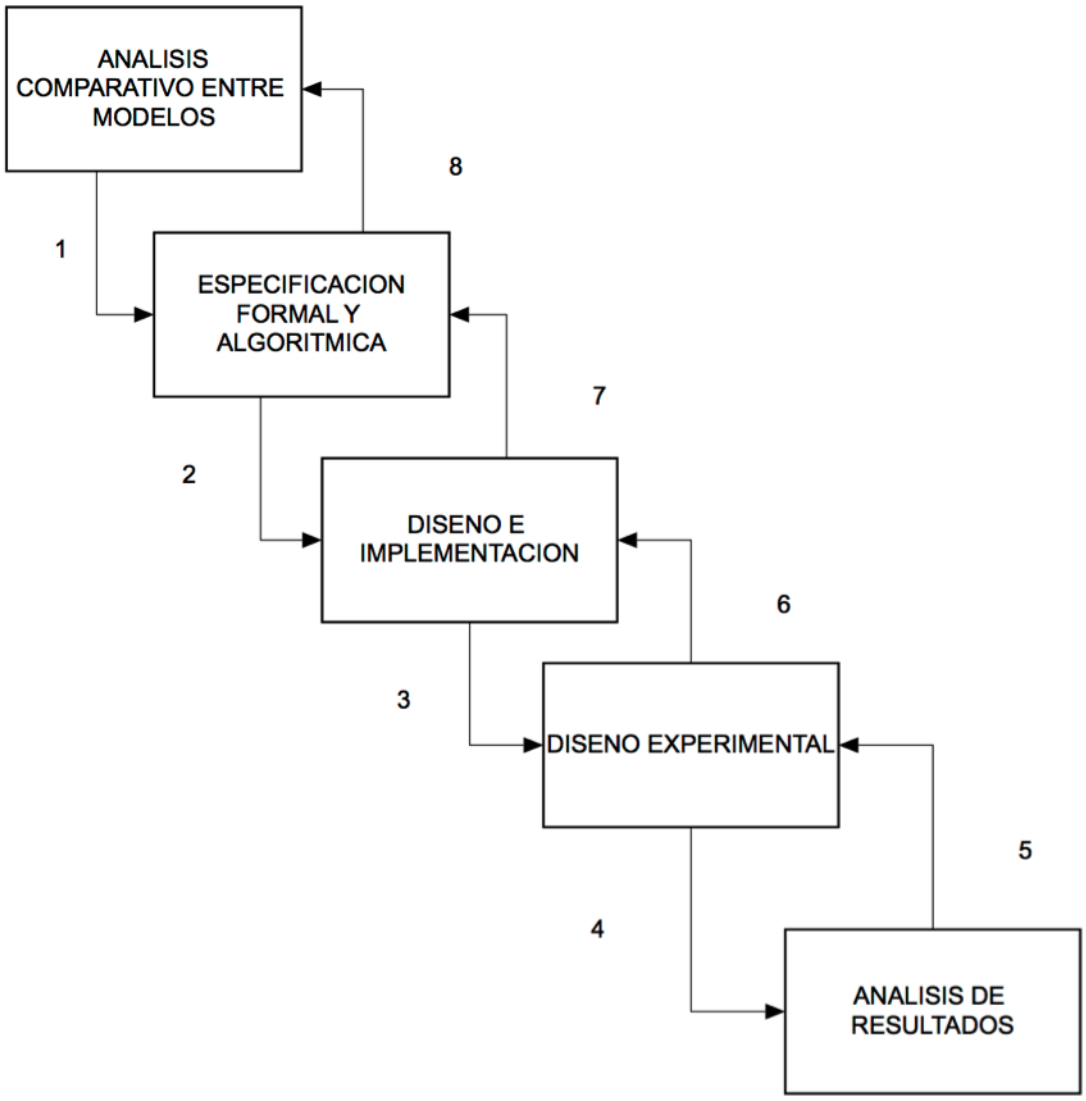


Figura 3.1: Metodología de la investigación

Capítulo 4

ANALISIS COMPARATIVO

A continuación, se presenta un análisis comparativo entre diferentes algoritmos que resuelven el control de concurrencia en el contexto de ambientes móviles. Según [21], [22] y [3], se pueden clasificar los algoritmos en los siguientes grupos: las estrategias pesimistas (ver Cuadro 4.1) y las estrategias optimistas (ver Cuadro 4.2), las estrategias conservadoras basadas en bloqueos y cerrojos,[23], [24] y las estrategias híbridas como la presentada en [3]. Esta última se refiere a una combinación de las características de las pesimistas y de las optimistas. Seguidamente se describen las dos primeras.

4.1. Control de concurrencia: estrategias pesimistas, optimistas y conservadoras en ambientes móviles

Las estrategias pesimistas trabajan bajo el siguiente principio "Do not allow any transaction to use the same shared data item, unless it is completed"[25]. En la estrategia pesimista, las transacciones son bloqueadas, hasta que esta no es exitosamente ejecutada, permitiendo así que los recursos puedan ser utilizados por transacciones que están en la cola de espera.

Uno de los factores importantes por contemplar en el contexto de los dispositivos móviles son las desconexiones frecuentes. Es por este motivo que es importante, para evitar el problema de *starvation*, que cada transacción tenga un tiempo asignado. Por lo tanto, si los resultados de las transacciones que se encuentran fuera de línea no son estipulados dentro de un tiempo

estimado, la transacción puede ser abortada y el recurso solicitado puede ser adquirido por otra transacción en espera.

Según [26], las estrategias pesimistas, tales como *Two Phase Locking* (2PL), requieren frecuentes intercambios de mensajes, reduciendo así el *thoughput* del sistema. Otro de los problemas que se puede presentar, es que inesperadas desconexiones pueden hacer que se indefinan los tiempos de bloqueo.

Otra de las estrategias empleadas en el contexto de los dispositivos móviles es la optimista. Esta estrategia trabaja el siguiente principio: "First perform operations, check for conflicts later"[25]. Es importante enfatizar que se han hecho estudios de esta corriente y se han creado modelos algorítmicos que han contribuido en el estado del arte del tema [27],[28], [29], [30], [31], [32]. Para los protocolos de *commit*, bajo esta estrategia el conflicto es tolerado en la fase de lectura. Sin embargo, si el sistema queda en un estado de inconsistencia, la técnica de resolución del conflicto se aplica en la fase de validación.

La estrategia optimista permite que muchos usuarios puedan leer al mismo tiempo datos que son compartidos. Cuando las transacciones llegan a *commit*, el recurso leído por otros usuarios ha sido invalidado. Para implementar esta técnica se utiliza una estrategia llamada *broadcasting*, la cual es utilizada sobre todo en países desarrollados donde hay tecnologías de comunicación satelital. Se caracteriza por ser costosa y de una implementación compleja.

Las políticas conservadoras (ver Cuadro 4.3) tienen como rasgo presentar bloqueos del recurso por utilizar. Algunos de los ejemplos usan la técnica de estampillas de tiempo o *timestamps*. Por otra parte, otros ejemplos de esta categoría de protocolos se basan en abortos, los cuales bajan la tasa de *commit* y de concurrencia.

4.1.1. Modelos pesimistas en ambientes móviles

A continuación, se analizan los principales algoritmos pesimistas según el enfoque dado por [25]:

- DHP-2L (Lam-2000)
- TCOT (Kumar-2002)
- Mobile Speculative Locking y Prioritized Concurrency Control (Reddy-1999)
- Prioritized Concurrency Control (Nizamuddin-2010)

Es importante rescatar que el análisis se hizo considerando las fortalezas y debilidades de cada uno de ellos.

Cuadro 4.1: Modelos pesimistas de control de concurrencia en ambientes móviles

Autor	**Modelos**	**Descripción**	**Fortalezas**	**Debilidades**
[33]	DHP-2L	Distributed real time locking protocol. Es un protocolo de bloqueo en tiempo real basado en HP-2PL.	Tiempo de espera depende de la prioridad de transacción.	Tipo de transacción no es anidada. Abortos de la transacción dependen de un *deadline*. Reinicia las transacciones tanto locales como globales.
[34]	TCOT	Trabaja sobre el principio de *time out*. Se busca el final del estado de la transacción en ejecución, evitando *starvation* del recurso compartido.	Tiempo de espera depende del máximo tiempo en ejecución. Tipo de transacción es combinada.	Abortos de la transacción dependen del valor de un umbral. Reiniciar la transacción si la ejecución no está completa en un límite establecido.
[35]	Mobile Speculative Locking	Se utiliza para reducir el bloqueo si se emplea 2PL	El *throughput* va de menos a más	Requiere de recursos extras y sigue una ejecución especulativa. Ocupa más capacidad de memoria.
[36],[37]	Prioritized Concurrency Control	Este protocolo se caracteriza por serializar las transacciones; si ocurriera algún conflicto toma en cuenta la transacción más antigua.	Serializa las transacciones y toma la más antigua.	Tipo de transacción no es anidada y el tiempo para volver a ejecutarse puede reducirse.. El *throughput* puede decrementar cuando las desconexiones aumentan.

4.1.2. Modelos optimistas en ambientes móviles

A continuación, se analizan los principales algoritmos óptimistas según el enfoque dado por [25]:

- OCC (Kung-1981)
- CCUD (Shirota-1999)
- CMM (Prabhu-2004)
- SODA (Xing-2008)

Es importante rescatar que el análisis se hizo considerando las fortalezas y debilidades de cada uno de ellos.

Cuadro 4.2: Modelos optimistas de control de concurrencia en ambientes móviles

Autor	**Modelos**	**Descripción**	**Fortalezas**	**Debilidades**
[6]	CCM	Distributed Real Time Locking Protocol. Es un protocolo de bloqueo en tiempo real basado en HP-2PL.	Tiempo de espera depende de la prioridad de transacción.	Tipo de transacción no es anidada. Abortos de la transacción dependen de un *deadline*. Reiniciar la transacción se hace en transacciones locales como globales.
[38]	SODA	Trabaja sobre el principio de *time out*. Se busca el final del estado de la transacción en ejecución, evitando *starvation* del recurso compartido.	Tipo de transacción es anidada y trabaja sobre el principio de *time out*.	Tiempo de espera depende del máximo tiempo en ejecución. Abortos de la transacción dependen de un valor del umbral. Reiniciar la transacción si la ejecución no está completa en un límite establecido.
[39]	CCUD	Protocolo que utiliza dos tipos de *broadcasting* uno por notificación, otro por certificación para invalidación de información.	Lanza un *broadcasting:* después de la validación local.	El *uplink bandwidth* incrementa exponencialmente cuando se comparten acceso entre datos.
[40], [41], [42],[7]	OCC	Protocolo libre de bloqueos o cerrojos; facilita el manejo de la computación móvil en relación con las desconexiones	Lanza un *broadcasting* de invalidación de reportes.	El recurso es solo de lectura y el *uplink bandwidth* es suficientemente bajo.

4.1.3. Modelos conservadores en ambientes móviles

A continuación, se analizan los principales algoritmos conservadores. Dentro de ellos están:

- FCU (Lim-1999)
- SGA (Lim-1999)
- ECHO (Shirota-1999)
- BUC (Al-Mogren-1999)

Es importante rescatar que el análisis se hizo considerando las fortalezas y debilidades de cada uno de ellos.

Cuadro 4.3: Modelos conservadores de control de concurrencia en ambientes móviles

Autor	**Modelos**	**Descripción**	**Fortalezas**	**Debilidades**
[1]	FCU	Forced Conflicts Under full Autonomy Protocol. Es un protocolo basado en políticas conservadoras.	Emplea *timestamp* para la serialización global. Se garantiza un orden que evita problemas entre conflictos directos e indirectos.	Está basado en políticas conservadoras tales como abortos. Presenta abortos y baja concurrencia.
[1]	SGA	Site Graph Algorithm Protocol. Es un protocolo basado en políticas conservadoras.	Emplea grafos no dirigidos.	Podría llegar a incurrir en ciclos y presenta baja concurrencia. Sufre de grandes *delays* por falsos abortos.
[11]	ECHO	Concurrency Control Method for a Large-scale Distributed Database. Es un protocolo basado en políticas conservadoras.	Utiliza la técnica de *Information Brodcasting*. Trabaja muy bien con gran cantidad de información.	Emplea bloqueos o cerrojos. Limita el número de sitios que se están actualizando. Cuando la cantidad de información es poca, no es tan eficiente ya que está basado en 2PL.
[12]	BUC	Simple yet Efficient Concurrency Control Technique for Mobile Data Broadcast Environment. Es un protocolo basado en políticas conservadoras.	Emplea *Broadcasting* y *Update cycles*.	No es tan eficiente en entornos pequeños o de procesamiento de pocos datos. Requiere de una arquitectura costosa económicamente, por lo que no es fácil de adquirir.

4.2. Comparación de los modelos clasificados

Como se ha mencionado, se clasificaron los modelos en tres categorías: la pesimista, la optimista y la conservadora. En relación con la pesimista, se pueden recuperar modelos donde se producen bloqueos, se reinician las transacciones, se trabaja bajo el principio de *time out*. En el caso de los optimistas, estos modelos presentan como características: se dan bloqueos en tiempo real, se emplean los *timestamps*, se trabaja bajo prioridad en la transacción, se trabaja bajo el modelo móvil planteado en secciones previas que incluye tanto transacciones locales como globales y se conceden tiempos de espera en relación con máximos tiempos de ejecución. Otra de las características sobresalientes es que se trabaja con la técnica de *broadcasting* cuya desventaja es que no es sencilla de implementar, pero para los ambientes móviles con desconexiones frecuentes ofrece un gran rendimiento.

A su vez, los modelos conservadores comparten entre otras características: *timestamp*, serialización global, establecimiento de estrategias de orden para evitar conflictos directos e indirectos. Se emplean grafos no dirigidos, se presentan ciclos y baja concurrencia. Otro aspecto por considerar de esta clasificación es que sufren *delays* y falsos abortos.

Al analizar las características anteriores, y por ser un ambiente móvil el que se plantea, es que no se tomarán en cuenta los bloqueos ni el reinicio de las transacciones en su totalidad de no ser posible llegar a *commit*. Aunque si es importante destacar que si se hubiera contado con una red inalámbrica satelital como en países desarrollados como Japón, China y EEUU, se hubiera integrado dentro de las características la técnica *Information Broadcasting*.

Tomando en cuenta todas estas fortalezas y debilidades citadas previamente y por el tipo de problema planteado en la presente investigación, es que se tomaron en cuenta las más favorables para el contexto móvil y las desconexiones frecuentes. Por lo tanto, el modelo propuesto puede ser considerado como híbrido.
Dentro de las características que tendrá la nueva propuesta están:

- Se implementó la aplicación utilizando el modelo cliente-servidor con el fin de poder utilizar diferentes tipos de escenarios distribuidos mediante un mismo gestor de base de datos, pero con varias instancias. Tanto el cliente como el servidor estuvieron representados con hilos (*threads*)

- Se utiliza el paradigma de orientación a objetos y las transacciones se implementaron como objetos (instancias de clase). Dando esto un mejor manejo de las validaciones, tanto del cliente como del servidor.
- No presenta bloqueos o cerrojos. Trabaja con un monitor implementado del lado del servidor que permite que se lleve a cabo cualquier tipo de transacción siempre y cuando no sea hacia un mismo esquema o tabla y que haya al menos una de esas transacciones que sea de escritura.
- No lleva a cabo abortos, ni reinicia recursos del sistema para atender las transacciones que tuvieron conflicto de concurrencia.
- Contempla las desconexiones frecuentes. Trabaja tres tipos de posibles desconexiones: por un problema de entrada y salida, por un problema de interrupción o bien por algún problema con el objeto transacción entre el cliente y el servidor.
- Simula el tiempo que puede durar un usuario en llevar a cabo todo el proceso transaccional con el dispositivo móvil. Lo anterior lo hace mediante una función aleatoria que calcula en milisegundos dicha actividad.
- Trabaja con estampillas de tiempo (*timestamp*) con el fin de evitar conflictos entre operaciones de escritura a un mismo esquema o tabla de una base de datos.
- No sufre de *delays* por no trabajar con abortos.
- No limita el número de estaciones móviles que se están actualizando.
- Tiende a mantener constante su desempeño, aún cuando la cantidad de transacciones se duplica.
- Es capaz de procesar considerable cantidad de transacciones (Ver Capítulo 7), específicamente en el análisis de resultados.
- No incurre en ciclos, por el tipo de solución basada en casos, resuelve las situaciones de conflicto.

A nivel de implementación, la propuesta se caracterizó por:

- Se implementó la aplicación utilizando el modelo cliente-servidor con el fin de poder utilizar diferentes tipos de escenarios distribuidos mediante un mismo gestor de base de datos, pero con varias instancias. Tanto el cliente como el servidor estuvieron representados con hilos (*threads*)

- Se utiliza el paradigma de orientación a objetos y las transacciones se implementaron como objetos (instancias de clase). Dando esto un mejor manejo de las validaciones, tanto del cliente como del servidor. El objeto transacción incluía: indicador del dispositivo, identificador de transacción, tipo de operación, esquema o tabla (*dataitem*),notificación y *timestamp*.

A continuación se le presentará al lector el planteamiento tanto del modelo formal o algorítmico como el modelo computacional propuesto en la investigación.

Capítulo 5

MODELO FORMAL Y COMPUTACIONAL

Con el propósito de una mejor comprensión y análisis del algoritmo, se ha hecho una caracterización en dos modelos: la especificación formal y algorítmica. En el modelo formal se dará la especificación matemática, mientras que en el computacional se explicará el algoritmo, así como cada una de sus instrucciones.

5.1. Modelo formal

Antes de definir los conceptos básicos del modelo, se establecerán varias definiciones previas necesarias para su entendimiento.

5.1.1. Sistema móvil

Definición 1:

Sea FH el servidor del sistema móvil propuesto y mh_w cada una de las estaciones de trabajo, se tiene que $MHS \in \{mh_1, mh_2, ..., mh_n\}$.

Definición 2:

Sea th_i, los *Threads* del servidor, se tiene que $FHS \in \{th_1, th_2, ..., th_m\}$ y cada uno de los th_i, es empleado para atender a los mh_w respectivos.

Definición 3:

Sea Th_j, los *Threads* de una estación móvil mhs_w, se tiene que $mhs_w \rightarrow th_j$ para atender cada una de las transacciones $T \in \{t_1, t_2, ..., t_x\}$.

5.1.2. *Fixed host*

Definición 4:

Sea FH la representación del servidor del modelo propuesto, se tiene que FH está conformado $FH = \{Socket, QueueTrans\}$ y estos a su vez están conformados $Socket = \{IP, Puerto\}$, $QueueTrans \in \{tg_1, tg_2, ..., tg_x\}$.

5.1.3. *Mobile host*

Definición 5:

Sea MH, la representación del dispositivo móvil del modelo propuesto, se tiene que $MH = \{Socket, QueueTrans, QueueWait, QueueDesconection\}$, donde $Socket = \{IP, Puerto\}$, $QueueTrans \in \{tl_1, tl_2, ..., tl_x\}$ y $QueueWait \in \{tl_1, tl_2, ..., tl_y\}$ y $QueueDesconection \in \{tl_1, tl_2, ..., tl_z\}$.

5.1.4. Transacción móvil

Definición 6:

Sea $MHS = \{mhs_1, mhs_2, ..., mhs_n\}$ el conjunto de n dispositivos móviles involucrados en un cierto escenario y $T = \{t_1, t_2, ..., t_m\}$ el conjunto de m transacciones que llevan a cabo cada uno de los dispositivos.

Se puede definir la función *t:* $T \rightarrow MHS$, donde $t(T_i) \subseteq MHS$. Ahora bien, si se aplica para varios dispositivos, se tendría $t(mhs_i)$ con $t(mhs_i) \subseteq MHS$ y por abuso, ver a $t(mhs)$ en vez de $t(\{mhs\})$.

Por otra parte, la definición formal de una transacción móvil se puede representar así:

Notación 1:

Sea $t(i) = \alpha a_1, ..., a_n \beta$ tal que
$t(i) = \alpha = begin$, $t_i(r) \in \{idTrans, idMH, dataItem, timeStamp, read, write\}$,
$tipOp \in \{read, write\}$ $\beta = commit$

Además, las operaciones *read* y *write* serán representadas como $tipOp_i$, no siendo un problema para el control de la concurrencia cuando el tipo de operación es solo lectura; es decir, $tipOp_i = read$.

Notación 2:

Se puede ver a t_i como una cadena de posibles operaciones donde se podría dar el problema de control de la concurrencia en casos como los siguientes:

- Si a_i tiene un $tipOp_i = read$ en el MH y $tipOp_j = write$ en el FH y la operación está en espera en el servidor.
- Si a_i tiene un $tipOp_i = write$ en el MH y $tipOp_j = write$ en el FH, y las dos sobre el mismo $dataItem$.
- Si a_i tiene un $tipOp_i = write$ en el MH y $tipOp_j = read$ en el FH, pero no la última versión actualizada.

Se debe tener en consideración que bajo el modelo planteado, solo se tendrá éxito por parte del móvil, en la medida en que β se pueda dar. Para que esto sea posible, es necesario que el móvil ejecute todas las transacciones que se encuentran pendientes pero siempre manteniendo la concurrencia tanto a nivel local como a nivel global.

5.1.5. Transacción local-global

Definición 7:

Sea t_g una transacción global y t_lp una transacción local, Se tiene que $t_{lp} \in T$. y $t_g = \{tl_1, tl_2, ..., tl_s\}$, donde ($tl_a \in t_g$ y $tipOp_i = write$) o $tl_a \in t_l$ y $tipOp \in \{read, write\}$.

5.1.6. Conflicto directo-indirecto

Seguidamente, se presentará la especificación formal tanto de conflictos directos como de conflictos indirectos:

5.1.7. Conflicto directo

Definición 8:

Sea t_i una transacción móvil y tiene un conflicto directo con t; y $dataItem$ un recurso determinado, se tiene $t_{(a)} = dataItem$ y $t_{(b)} = dataItem$ y pueden tener conflictos de control de concurrencia:

- Si $\exists$ b t.q. $t_{i(b)} = dataItem$, antes de que se dé $t_{i(a)} = dataItem$.

5.1.8. Conflicto indirecto

Definición 9:

Sean dos transacciones t_a y t_b; se dice que t_a y t_b presentan un conflicto indirecto, si existen $t_1, ..., t_n$ transacciones tales que t_a tiene conflicto directo con t_b y t_i con t_{i+1} con:
$i = 1, ..., n-1$. Se dice que un conflicto indirecto se presenta cuando:

- $t_{a(x)}, t_{b(x)}$.
- t_a tiene conflicto directo con t_1 y t_1 tiene conflicto con $t_2, ..., t_n$ y este con t_b.

Considerando que cada elemento del conjunto Y es visto como una transacción $t(y_i)$ y puede ser de índole directa, es decir, controlada por un DBMS local o bien de índole indirecta donde la transacción forma parte tanto del sistema local como del sistema global, afectando así el conocimiento que tenga la historia global de la local.

5.1.9. Desconexión frecuente

Definición 10:

Sea DF una desconexión frecuente; esta puede darse ya sea por parte de MH_i o FH, es decir, por parte del dispositivo móvil o bien del servidor respectivamente.

Notación 3:

Se puede ver a D como una condición de desconexión relacionada con un estado $W \in \{IE, IOE, CNotFE\}$ del sistema, el cual para cumplirse debe darse lo siguiente:

- Si $W = \{IE, IOE, CNotFE\}$ entonces MH_i o FH sufrió una desconexión del sistema y $D \in \{BackupDB\}$
- Sino entonces MH_i o FH no sufrió una desconexión del sistema y $D \notin \{BackupDB\}$

5.1.10. Latencia de la red

Definición 11:

Sea δ el tiempo representado por la latencia de la red, entonces δ está conformado por valores discretos y $\delta \in \{1, 2, ..., 5000\}$ correspondientes a los milisegundos que puede durar el servidor en atender una transacción proveniente de algún mhs_i.

5.2. Modelo algorítmico

El modelo computacional consiste en dos algoritmos los cuales representan tanto el lado del cliente (*Mobile Host*) como el lado del servidor (*Fixed Host*). Para la construcción de los algoritmos, se tomó como referencia la investigación hecha en [9],[43].
Como se ha mencionado en previos apartados, los dos tipos de transacciones básicas por representar en el algoritmo serán:

- Transacciones globales.
- Transacciones locales.

Las transacciones globales, estarán representadas por un agente llamado *serverQueue*, el cual se manejará dentro del servidor *(Fixed Host)*. Otra estructura similar será administrada por parte del cliente *(Mobile Host)* llamada *clientQueue*.
Cuando se habla de una transacción, se refiere a una estructura que tiene la siguiente forma:

- *idTrans*
- *idMH*
- *dataItem*
- *timeStamp*
- *tipOp*

El primer parámetro de la estructura anterior se refiere al identificador de la transacción que será ejecutada por parte de alguna estación remota o *Mobile Host*. Seguidamente, se presenta el identificador de *Mobile Host (id-MH)* el cual representa un valor único y exclusivo que distingue el dispositivo móvil. Por su parte, en relación con el *dataItem*, corresponde al objeto al cual se le aplicará la operación que puede ser como se ha descrito en el modelo formal, una operación de escritura o bien de lectura. Dicha operación está representada por el parámetro *tip-Op*. En el caso del *timeStamp*, corresponde a una estampilla de tiempo con un valor único, necesaria para diferenciar la transacción del resto en cuanto a su tiempo de llegada.

5.2.1. Algoritmo cliente (*Mobile Host*)

```
1: Inicializate queueTrans, queueWait y queueDesc
2: CreateTransactions
3: if backupDB.getQueueDesconex()==null then
4:    Sin Transacciones desconectadas
5: else
6:    Transacciones desconectadas
7: end if
8: while !queueDesc AND !queueWait AND !queueTrans do
9:    read Transaction
10:   if connectFixed-Host then
11:      createSocket
12:      writeTransaction
13:      if TransacctionNotif then
14:         ExecuteTransaction
15:         print Successful Transaction
16:         Noconflict = Noconflict + 1
17:      else
18:         NoExecuteTransaction
19:         print Conflict Transaction
20:         add QueueWait
21:         conflict = conflict + 1
22:      end if
23:   else
24:      Exception
25:   end if
26:   sleep(RandomNumber (5000))
27:   closeSocket
28:   if IOException then
29:      BackupDB
30:   end if
31:   if InterruptedException then
32:      BackupDB
33:   end if
34:   if ClassNotFoundException then
35:      BackupDB
36:   end if
37: end while
38: print Tiempo de Commit
39: print Transacciones con Conflicto
40: print Transacciones sin Conflicto
```

Algoritmo 1: Cliente (*Mobile Host*)

Dentro de las principales secciones del algoritmo Cliente están:

- Una primer sección para verificar si existió desconexión y procesarlas (backupDB).
- Se maneja un ciclo contemplando las tres colas utilizadas en el modelo: desconexión, espera y transacciones.
- Una vez dentro del ciclo, se procede a enviar la transacción al servidor y a contabilizar si ésta tuvo o no tuvo conflicto.
- Si la notificación del servidor es positiva se lleva a cabo el *commit*, de lo contrario se vuelve a ejecutar sin reiniciar ni reservar recursos extras.
- Cuando ya se hayan procesado las colas, se procede al reporte de estadísticas: tiempo de commit, transacciones sin conflicto y transacciones con conflicto

5.2.2. Algoritmo servidor (*Fixed Host*)

```
1: Inicializate queueTrans
2: while TRUE do
3:    create Socket
4:    read TIn
5:    TIn.setTimeStamp(TRUE)
6:    TIn.setTimeStamp(ts)
7:    ts = ts + 1
8:    queueTrans.add(TIn)
9:    queueTransAux = queueTrans
10:   while queueTransAux AND queueTransAux.size > 1 do
11:      TOut = queueTransAux.poll
12:      if TIn.IdMH()=TOut.IdMH() AND
         TIn.DataItem()=TOut.DataItem() AND TOut.tipOp()=W then
13:         if TIn.tipOp()=R then
14:            TIn.setNotif(FALSE)
15:         end if
16:         if TIn.tipOp()=W then
17:            if TIn.TimeStamp() > tOut.TimeStamp() then
18:               TIn.setNotif(FALSE)
19:            end if
20:         end if
21:      end if
22:      if TIn.IdMH()=TOut.IdMH() AND
         TIn.DataItem()=TOut.DataItem() then
23:         queueTransAux.remove(TIn)
24:      end if
25:   end while
26: end while
```

Algoritmo 2: Servidor (*Mobile Host*)

Dentro de las principales secciones del algoritmo Servidor están:

- Trabaja una cola de transacciones que corresponden a las transacciones globales (monitor transaccional). Este monitor será el encargado a de aplicar las reglas de control de concurrencia.
- Trabaja un ciclo infinito para procesar toda la cantidad de clientes que sean aceptados por el *socket*.
- Se analizan los posibles escenarios donde pueda haber problema de control de concurrencia.
- Las comparaciones se hacen entre la transacción entrante y las transacciones que se encuentran en la cola del servidor
- Se le notifica al cliente respecto a si puede o no llevar a cabo la transacción (llevarla hasta *commit*)

A continuación se le presentará al lector el planteamiento del diseño experimental. Sección que permitirá tener mayor claridad en relación a la validación de los resultados obtenidos del simulador implementado.

Capítulo 6

DISEÑO EXPERIMENTAL

Seguidamente, se procederá a explicar en forma detallada cómo fue que se hizo la experimentación de la investigación propuesta.

6.1. Implementación de un simulador

Con el objetivo de validar la nueva propuesta algorítmica planteada en el capítulo 5, se llevó a cabo un simulador en el lenguaje de programación Java, específicamente en la plataforma Android. Es importante destacar que tanto los dispositivos móviles *(Host)* como el servidor*(Fixed)* están representados por hilos(*threads*) para poder trabajar la programación de forma concurrente. El simulador cuenta con dos clases: el *Mobile Host* (MH) y el *Fixed Host* (FH).

El principal objetivo de este apartado ha sido constatar que la propuesta computacional logra resolver el problema de control de concurrencia.

Dentro de la simulación, las operaciones relacionadas con escribir o con leer sobre una base de datos fueron representadas mediante la simbología *write* (W) y *read* (R).

6.2. Escenarios del simulador

Los escenarios que fueron definidos tuvieron en común los siguientes elementos:

- *Web service*

  Representa el hilo servidor, el cual, como se ha mencionado, está esperando las solicitudes de los hilos clientes los cuales se conectarán a este para mandarle la transacción y de esta manera poder comparar si la transacción se encuentra o no dentro del contexto de la historia global.

- Una configuración de dispositivos que varía según el escenario

  La configuración aumenta según sea el escenario, con el fin de probar el simulador con distinta cantidad de transacciones y dispositivos. Se fue haciendo una exigencia cada vez mayor del algoritmo propuesto, conforme se iban probando cada uno de los cinco escenarios planteados.

- La forma en que fueron obtenidas las transacciones

  Como se trata de una simulación, la generación del orden de las transacciones se hace mediante una función de aleatoriedad, con el fin de cumplir con uno de los principios básicos de un experimento citado en [44], donde se establece que la aleatoriedad es uno de los principales elementos de un experimento y que lo distingue de un cuasi-experimento o caso de estudio.

.
Es importante rescatar, que las desconexiones fueron simuladas dentro de cada uno de los escenarios expuestos en el Cuadro 6.1 y corresponderá a un rango de un mínimo de 1 y un máximo de 5 desconexiones la(s) cual(es) será(n) calculada(s) por medio de una función aleatoria en el lenguaje de programación Java.

Cuadro 6.1: Escenarios del sistema móvil simulado

Escenario	**Configuración**	**Generación transaccional**
E1	*Web service*, dos dispositivos móviles (2 celulares) y 200 transacciones (escrituras-lecturas)	Aleatoria
E2	*Web service*, dos dispositivos móviles (2 celulares) y 400 transacciones.	Aleatoria
E3	*Web service*, dos dispositivos móviles (2 celulares) y 800 transacciones.	Aleatoria
E4	*Web service*, tres dispositivos móviles (3 celulares) y 1200 transacciones.	Aleatoria
E5	*Web service*, tres dispositivos móviles (3 celulares) y 2400 transacciones.	Aleatoria

6.3. Definición de métricas

Esta sección define cada una de las métricas que fueron empleadas para el análisis de resultados:

Cuadro 6.2: Métricas para comparar los algoritmos

Métrica	**Descripción**	**Unidad de medida**
Tiempo de *commit*	Se refiere al tiempo que tardará el dispositivo en llegar a *commit*	Segundos
Complejidad algorítmica	Se lleva a cabo un análisis de la complejidad algorítmica del algoritmo propuesto	Omega
Tasa de conflictos de concurrencia por escenario	Se refiere a una transacción ejecutada con conflicto en la concurrencia	Porcentaje de conflictos
Tasa de transacciones sin conflictos de concurrencia por escenario	Se refiere a una transacción ejecutada sin conflicto en la concurrencia	Porcentaje de conflictos

6.4. Comparación entre algoritmos

Tomando en cuenta la definición de las métricas anteriores, se llevó a cabo una comparación entre el modelo propuesto y un algoritmo conocido como [3], analizado en el Capítulo 3 de la investigación propuesta. Se seleccionó este estudio, principalmente por ser uno de los algoritmos analizados, por

la facilidad de acceso al algoritmo que utilizaron, por seguir la arquitectura cliente-servidor, por ser reciente (año 2008), por se un algoritmo híbrido y por que era el algoritmo con características más fáciles de adaptar a los escenarios de prueba de la presente propuesta de investigación. Es importante dejar claro que ambos algoritmos fueron corridos bajo estructuras semejantes con el fin de mantener la validez del experimento y de la comparación.

Seguidamente se presenta el algoritmo tanto a nivel del cliente como a nivel del servidor de la propuesta hecha en [3] e inspirado y basado en otros estudios [45], [46], [35], [45], [47], [48], [49], [50]:

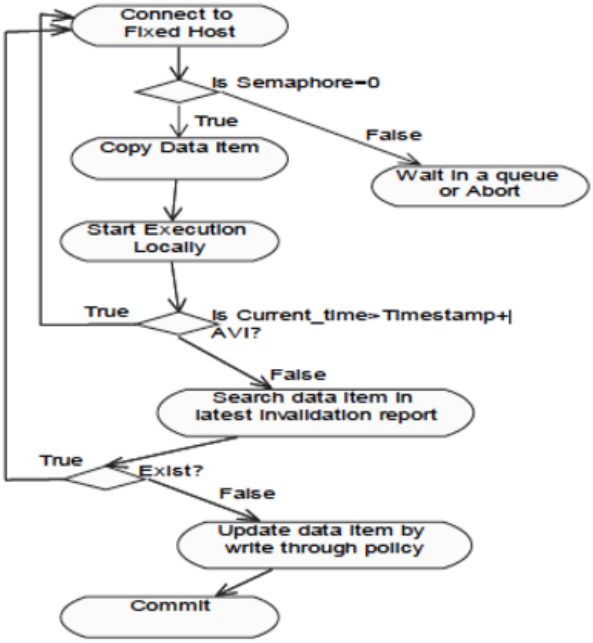


Figura 6.1: Algoritmo del Cliente en Moiz-Algorithm[3]

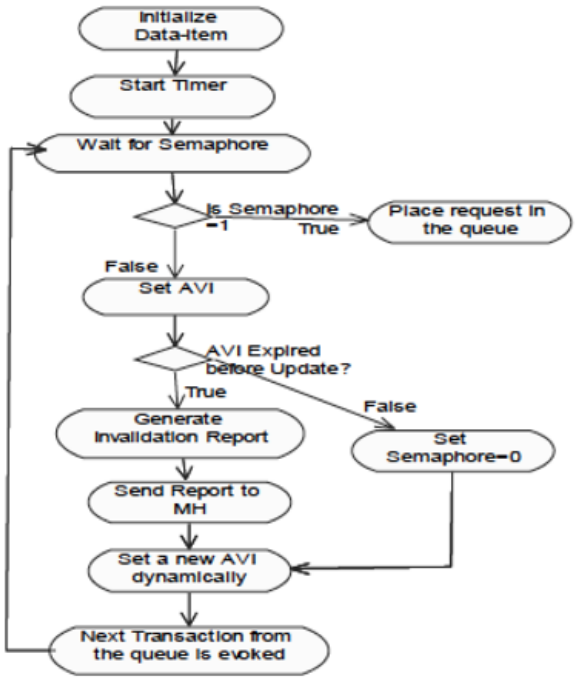


Figura 6.2: Algoritmo del Servidor en Moiz-Algorithm[3]

6.5. Diseño de análisis estadístico

El análisis estadístico se siguió tomando como referencia [44] como literatura base; sin embargo se tomaron en cuenta otras referencias tales como: [51], [52], [53], [54]. Dentro de los principales apartados están: la variable de respuesta del estudio, la unidad experimental, el orden de la corrida, la recolección de los datos, el modelo estadístico, el análisis de los resultados y las limitaciones del diseño.

6.5.1. Variable de respuesta del estudio

Como variable del estudio se empleó el tiempo promedio de *commit* en los algoritmos estudiados en relación con los cinco escenarios planteados. Este tiempo se obtuvo sacando el promedio del tiempo de *commit* que se da entre cada dispositivo. Según sea el escenario, este promedio es el resultado de dividir la suma de los tiempos de *commit* de cada dispositivo, según el caso entre la cantidad de dispositivos.

La variable de respuesta correspondió a la transacción y su representación interna fue:

Cuadro 6.3: Estructura de una transacción

Transacción	**Descripción**
IdTransacción	Corresponde a la identificación de la transacción
idMovil	Corresponde a la identificación del dispositivo
dataItem	Corresponde al esquema (tabla) por consultar o modificar
tipOp	Corresponde al tipo de transacción por ejecutar (*Write* o *Read*)
notif	Se notifica verdadero o falso al ejecutar la transacción

6.5.2. Unidad experimental

En relación con la unidad experimental, esta estuvo conformada por las operaciones tanto *Write* como *Read* generadas por parte del simulador.

6.5.3. Orden de la corrida

El orden de corrida se hizo de manera aleatoria para seguir uno de los principios básicos del diseño experimental [44]. El orden de la corrida del experimento fue:

Cuadro 6.4: Orden de la corrida de las observaciones

Escenario	**JD-Algorithm**	**Moiz-Algorithm**
E1	10	3
E2	1	9
E3	7	2
E4	4	6
E5	8	5

6.5.4. Recolección de los datos

La recolección de los datos se hizo en forma simulada, es decir que la representación de las transacciones fue hecha por parte de un simulador programado en Java específicamente en Android. Como entradas se utilizaron tanto *Write* como *Read*. Otro de los aspectos importantes es que se utilizó el mismo esquema o tabla de la base de datos (la tabla clientes) con el fin de simular los posibles escenarios de conflicto de control de concurrencia entre los dispositivos.

6.5.5. Modelo estadístico

El modelo estadístico utilizado fue el de efectos fijos según [44, p. 141]:

$Y_{ijk} = \mu + \tau_i + \beta_j + \epsilon_{ijk}$, $i = 1...a$ y $j = 1...b$, $k = 1...r$ donde:

- μ es la media poblacional de los escenarios: Escenario-1, Escenario-2, Escenario-3, Escenario-4 y Escenario-5.

- τ_i es el efecto del tratamiento i-esímo. Dentro de los tratamientos están: Escenario-1, Escenario-2, Escenario-3, Escenario-4 y Escenario-5.
- β_j es el efecto del bloque j-esímo. Dentro de los bloques están: JD-Algorithm, Moiz-Algorithm.
- ϵ_{ijk} es el usual NID(0,σ^2).

6.5.5.1. El Anova

El Anova que se empleó fue el correspondiente al modelo *Randomized Complete Block Design*(RCBD). Las razones por la que se utilizó este Anova fueron:

- Como se ha mencionado, la simulación que se llevó a cabo hizo uso de aleatorización no solo por el orden de la corrida indicado en el Cuadro 6.4 , sino también por la forma de calcular las transacciones *Write(W)* y *Read(R)*.
- Se utilizaron todos los tratamientos, es decir los niveles del factor escenario, por lo que se empleó un diseño completo.
- Se utilizaron bloques (los algoritmos), con el fin de estudiar el comportamiento de los usuarios pero por bloques o grupos homogéneos.

Es importante contextualizar algunos términos. Así por ejemplo, el uso de a para tratamientos, es decir los escenarios: Escenario-1, Escenario-2, Escenario-3, Escenario-4 y Escenario-5 y b para bloques: JD-Algorithm, Moiz-Algorithm. [44, p. 146]

6.5.5.2. El criterio de rechazo

El criterio de rechazo de H_0 correspondiente para RCBD fue:

- $f_0 > f_{\alpha,(a-1),(b-1)}$

  Para el caso del problema, si se da que $f_0 > f_{\alpha,(a-1),(b-1)}$ entonces se rechaza H_0, lo cual significará que existe al menos una media aritmética de algún escenario de los propuestos que no fue igual. Por otra parte, si H_0 no se rechaza, significa que todos los escenarios tuvieron la misma

media aritmética y para efectos del estudio significaría que todos los algoritmos tuvieron el mismo tiempo promedio de *commit*.

Es importante y transcendental tener en cuenta qué significa cada valor de la expresión matemática. Así por ejemplo, el f_0 representa el valor de *Fisher* para el estudio. Otro aspecto importante es el valor del nivel de significancia α, el cual normalmente es 0,05 y por medio de él se puede obtener el nivel de confianza para rechazar o no rechazar H_0. En un caso como $\alpha = 0{,}05$, el nivel de confianza sería de 95 por ciento de confianza estadística de rechazar la hipótesis.

6.5.5.3. La hipótesis de la investigación

La hipótesis de la investigación es probar que existe diferencia significativa entre el algoritmo propuesto (JD-Algorithm) y el algoritmo estudiado (Moiz-Algorithm) y si es así analizar la diferencia producida entre algoritmos.

6.5.5.4. La hipótesis estadística

Seguidamente, se presenta la hipótesis estadística:

$H_0 : \mu_1 = \mu_2 = \mu_3 = \mu_4 = \mu_5$
$H_1 : \mu_i \neq \mu_j$, al menos un $i \neq j$

Cada μ_i representa la media aritmética de cada escenario. Con esta hipótesis estadística, se validó la igualdad de las medias de los diferentes escenarios del sistema, tomando en cuenta los dos bloques (algoritmos) utilizados.

Según la hipótesis planteada previamente, cada μ_i representó la media aritmética de cada uno de los algoritmos estudiados.

6.5.6. Análisis de resultados

El análisis de resultados se dio mediante *boxplots* para poder analizar los percentiles de cada uno de los algoritmos. Por otra parte, en cuanto a los *tests* estadísticos que se aplicarán están:

- Test de normalidad

  Se pretende probar la normalidad de la población de datos, desde el punto de vista estadístico. Para esto se recomienda utilizar el *Test*

de Shapiro-Wilk utilizado en varios estudios tales como [55], [56], [57] para poblaciones menores a 30 y para mayores se utiliza el *Test de Kolmogorov-Smirnov* [56],[58].

- Test de igualdad de varianzas

  Se validan las varianzas, las cuales con respecto a la teoría de probabilidades se refieren a una medida de dispersión que representa las diferencias con la media elevadas al cuadrado. Se suelen hacer algunos *tests* para comprobar la igualdad. Sin embargo, para efectos de la investigación se utilizó el *Test de Bartlett* el cual ha sido ampliamente aplicado [59],[60],[61]. Otro estudio que se aplica para verificar la igualdad de varianzas es Levene Test [62], [63], [64]. Este se caracteriza por ser no paramétrico, es decir más sensible a la distribución normal de los datos por estudiar. Mientras que el utilizado en el presente estudio (Bartlett) es paramétrico, o sea menos sensible a la distribución normal de los datos por estudiar.

- Independencia

  Se valida el principio de independencia presente en la teoría de probabilidad. Este principio establece que dos sucesos aleatorios son independientes entre sí cuando la probabilidad de cada uno de ellos no está influenciada por que el otro suceso ocurra o no, es decir, cuando ambos sucesos no están relacionados. En el caso de modelo propuesto al no seguir un esquema anidado, cada transacción adquiere independencia en sí misma aunque compartan en común un mismo esquema o tabla.

Seguidamente, se detallan los resultados de la propuesta experimental planteada en este capítulo.

Capítulo 7

ANALISIS DE RESULTADOS

Esta sección tiene como principal objetivo mostrarle al lector el desempeño obtenido por el algoritmo propuesto desde el punto de vista de las métricas definidas en el Capítulo 6 de la presente propuesta de investigación. Para la evaluación y análisis de los resultados, se han tomado como referencia dos posibles escenarios comparativos. El primer escenario corresponde a un *benchmarking* con una propuesta conservadora [3]. El segundo escenario corresponde a un análisis del algoritmo en cuanto a su desempeño. Para ello se tomó en cuenta: tiempo de *commit*, complejidad algorítmica, transacciones conflictivas y transacciones sin conflicto.

7.0.7. Análisis de la comparación entre algoritmos

Seguidamente se le presenta al lector en el Cuadro 7.1 un resumen del tiempo promedio de *commit* (TPC), tanto del algoritmo propuesto como del algoritmo conservador.
Para la comprobación de la idoneidad del modelo estadístico se llevaron a cabo los siguientes *tests* y *plots* estadísticos:

1. Comprobación de normalidad

   En la Figura 7.1, se obtuvo que los residuos del experimento son normales, ya que el valor de p en este caso fue de 0,0515, valor que es mayor al $\alpha = 0{,}05$ por lo tanto no se rechaza H_0. Al no rechazar H_0 significa que existe normalidad en los datos. Por lo tanto, se puede concluir que se está trabajando con una población estadísticamente normal El resultado obtenido en R se muestra en la Figura 7.1.

Cuadro 7.1: Tiempo promedio de *commit* por escenario (segundos)

Escenario	**Móviles**	**Transacciones**	**JD-Algorithm**	**Moiz-Algorithm**
E1	2	200	376	778
E2	2	400	684	1150
E3	2	800	1537	3400
E4	3	1200	1593	4550
E5	3	2400	3170	6520

```
> # Estudio de Normalidad
> print(shapiro.test(dat.lm$residuals)) # normalidad

    Shapiro-Wilk normality test

data:  dat.lm$residuals
W = 0.8456, p-value = 0.0515
```

Figura 7.1: Shapiro para normalidad

Por otra parte, para poder visualizar la normalidad se empleó un *plot* de normalidad en R llamado *qqplot*, el cual se muestra en la Figura 7.2. Según el comportamiento de las observaciones en esta gráfica y por la distribución que presentan las observaciones, existe un comportamiento de normalidad que ratifica conjuntamente con el *Test de Shapiro* el hecho de que la población tiende a la normal, principio de gran importancia para poder llevar a cabo el estudio de Anova [65]. Por otra parte, es importante señalar la presencia de dos atípicos o valores conocidos como *outliers*. Este tipo de observaciones puede ser interesante para posibles análisis.

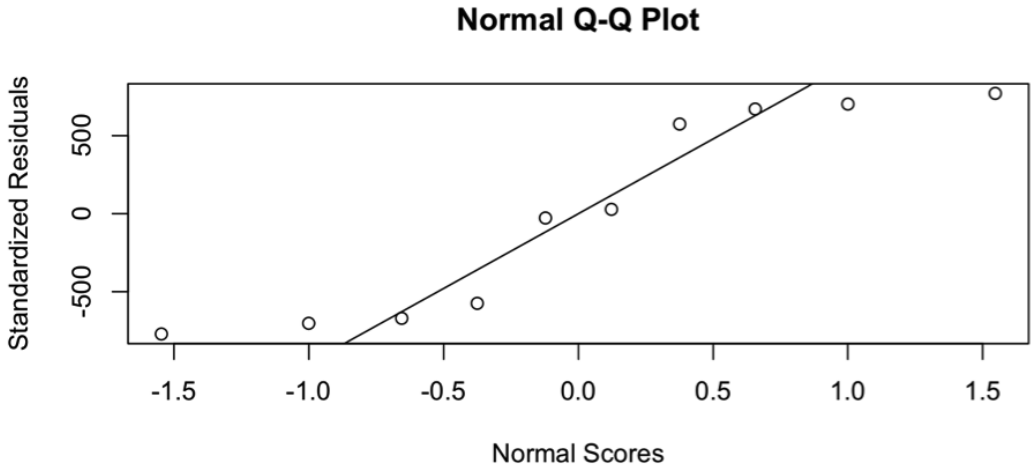


Figura 7.2: Gráfica para normalidad

2. Comprobación de igualdad de varianzas

   Seguidamente, se muestra en la Figura 7.3 cómo el valor de $p = 0{,}1553 > 0{,}05$, por lo que se puede asegurar igualdad de varianzas según el test de Bartlett. En la Figura 7.3, se presenta la prueba de igualdad de varianzas hecha en R para el bloque algoritmo:

```
> # Estudio de Igualdad de varianzas Algoritmo
> print(bartlett.test(TPC ~ Algoritmo, data=dat)) # Varianzas

    Bartlett test of homogeneity of variances

data:  TPC by Algoritmo
Bartlett's K-squared = 2.0191, df = 1, p-value = 0.1553
```

Figura 7.3: Bartlett para igualdad de varianzas(Algoritmo)

A su vez, en la Figura 7.4, correspondiente a los escenarios, el valor de $p = 0{,}4334 > 0{,}05$ comprobando que sí existe igualdad de varianzas. Es importante rescatar que aunque en ambos casos (algoritmos y escenarios) existe igualdad de varianzas, hubo un valor más alto en el caso del p_{value} de los escenarios con respecto a los algoritmos, dándose una diferencia de $p = 0{,}2781$

```
> # Estudio de Igualdad de varianzas Escenarios
> print(bartlett.test(TPC ~ Escenario, data=dat)) # Varianzas

    Bartlett test of homogeneity of variances

data:  TPC by Escenario
Bartlett's K-squared = 3.8026, df = 4, p-value = 0.4334
```

Figura 7.4: Bartlett para igualdad de varianzas(Escenarios)

Con el fin de poder visualizar la igualdad de varianzas, se muestra en la Figura 7.5 la comprobacíon. La forma de interpretarla es mediante la no aparición de patrones en los residuos indicando así igualdad de varianzas cuando no hay patrones claros y establecidos.

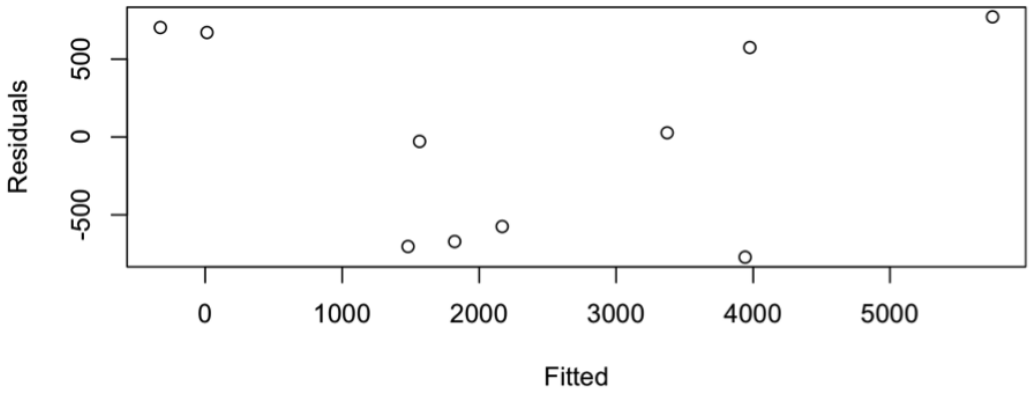


Figura 7.5: Gráfica para igualdad de varianzas

3. Supuesto de independencia

   En teoría de probabilidades, se dice que dos sucesos aleatorios son independientes entre sí cuando la probabilidad de cada uno de ellos no está influida porque el otro suceso ocurra o no, es decir, cuando ambos sucesos no están relacionados. Con base en lo anterior y dado que el presente experimento fue hecho de forma simulada utilizando la aleatoriedad, cada observación obtenida fue calculada totalmente independiente respecto a las otras.

   De ahí, que según este principio de probabilidades se puede asumir independencia de las observaciones de las muestras.

Sobre la base de los estudios anteriores (normalidad, igualdad de varianzas e independencia), se puede aplicar el Anova en estudio.

En la Figura 7.6 se muestra el Anova del experimento basado en el Cuadro 7.1 de datos obtenidos, producto de los resultados de los algoritmos implementados:

```
> # Modelo Lineal
> dat.lm=lm(TPC ~ Escenario + Algoritmo, data=dat)
> anova(dat.lm)
Analysis of Variance Table

Response: TPC
           Df   Sum Sq Mean Sq F value  Pr(>F)
Escenario   4 23906639 5976660  6.3932 0.04994 *
Algoritmo   1  8168544 8168544  8.7378 0.04172 *
Residuals   4  3739395  934849
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Figura 7.6: Anova de la simulación

Dentro de los aspectos más relevantes estuvieron:

1. El rechazo de H_0 ya que el $p = 0{,}04994$ fue menor al $\alpha = 0{,}05$, lo que demuestra que sí existe diferencia de medias, es decir que las medias de los tratamientos (escenarios) no son todas iguales. O sea, existe al menos un par de ellas que difieren.

2. Respecto al valor de $p = 0{,}4172$ en el caso del bloque (algoritmo), se puede interpretar que al igual que en el caso anterior, sí existieron diferencias significativas entre las medias de los algoritmos JD-Algorithm y Moiz-Algorithm. Lo anterior se debió a que el valor de $p = 0{,}14457$ es menor a $\alpha = 0{,}05$ por lo que hay rechazo de H_0 lo cual quiere decir que el algoritmo propuesto tuvo un comportamiento distinto del algoritmo conservador caracterizado por no llevar a cabo cerrojos sobre los mismos esquemas y no reiniciar recursos para llevar a cabo transacciones que estuvieron en conflicto.

Al ser rechazada la hipótesis nula, se puede inferir que no todas la medias aritméticas tanto de los tratamientos como de los bloques son iguales. Es decir, existe al menos un par de medias diferente tanto en los tratamientos como en los bloques.

Con el fin de verificar cuál o cuáles de esas medias fueron iguales o diferentes, se aplicó un estudio de *Fisher LSD* fundamentado en [66], para descubrir el par o los pares de medias que difirieron y en cuánto difirieron.

A continuación en la Figura 7.7, se muestra el resultado en R del estudio de *Fisher LSD*:

```
> print(LSD.test(d,"Escenario"))
$statistics
    Mean       CV  MSerror      LSD
  2375.8 40.69685 934848.7 2684.477

$parameters
  Df ntr  t.value
   4   5 2.776445

$means
     TPC       std r         LCL      UCL  Min  Max
1  577.0  284.2569 2 -1321.2121 2475.212  376  778
2  917.0  329.5118 2  -981.2121 2815.212  684 1150
3 2468.5 1317.3399 2   570.2879 4366.712 1537 3400
4 3071.5 2090.9148 2  1173.2879 4969.712 1593 4550
5 4845.0 2368.8077 2  2946.7879 6743.212 3170 6520

$comparison
NULL

$groups
  trt  means  M
1   5 4845.0  a
2   4 3071.5 ab
3   3 2468.5 ab
4   2  917.0  b
5   1  577.0  b
```

Figura 7.7: Fisher LSD

Según la Figura 7.7, los pares de medias significativamente diferentes son aquellos que no poseen la misma letra y esto se puede calcular como se muestra en el Cuadro 7.2.

Cuadro 7.2: Pares de medias significativas según Fisher LSD

(ij)	**abs($y_i - y_j$)**	**Relación**	**Dif. Significativa**
1-2	abs(577.0-917.0)= 340	340 < 2684,4	No
1-3	abs(577.0-2468.5)=1891.5	1891,5 < 2684,4	No
1-4	abs(577.0-3071.5)= 2494.5	2494,5 < 2684,4	No
1-5	abs(577.0-4845.0)=4268	4268 > 2684,4	Sí
2-3	abs(917.0-2468.5)= 1551.5	1551,5 < 2684,4	No
2-4	abs(917.0-3071.5)=2154.5	2154,5 < 2684,4	No
2-5	abs(917.0-4845.0)= 3928	3928 > 2684,4	Sí
3-4	abs(2468.5-3071.5)=603	603 < 2684,4	No
3-5	abs(2468.5-4845.0)=2376.5	2376,5 < 2684,4	No
4-5	abs(3071.5-4845.0)=1773.5	1773,5 < 2684,4	No

Basado en el Cuadro 7.2, se puede inferir:

1. El par de medias con mayor diferencia significativa fue 1 y 5 correspondiente a 4268 segundos.

2. El par de medias que tuvo la diferencia significativa más baja fue 1 y 2, presentando una diferencia de 340 segundos.

3. El escenario con mayor media fue el 5 (2400 transacciones) con 4845,0 segundos y con menor media fue el escenario 1 con 577,0 segundos.

A continuación, se muestra en la Figura 7.8 el *boxplot* correspondientes a los bloques del estudio (algoritmos).

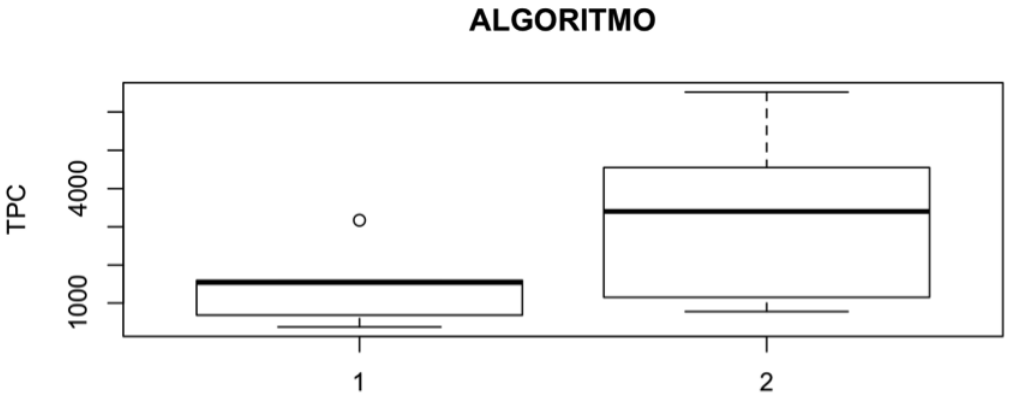


Figura 7.8: *Boxplot* por algoritmo

De la Figura 7.8 tomando en cuenta los resultados de la herramienta estadística R, se puede decir del primer algoritmo (propuesto), mientras que el segundo corresponde al algoritmo conservador, que presenta un $n = 5$ correspondiente a los cinco escenarios planteados y explicados en capítulos previos.

1. Algoritmo-1 (JD-Algorithm) tuvo un valor mínimo de 376 segundos, un percentil 25 de 684 segundos, un percentil 50 o sea la mediana de 1537 segundos y un percentil 75 con un valor máximo de 1593 segundos. A su vez, presenta una observación *outlier* con un valor de 3170 segundos.

2. Algoritmo-2 (Moiz-Algorithm) tuvo un valor mínimo de 778 segundos, un percentil 25 de 1150 segundos, un percentil 50 o sea la mediana de 3400 segundos, un percentil 75 con un valor de 4550 segundos y un valor máximo de 6520. A su vez, presenta una observación *outlier* con un valor de 3170 segundos.

Seguidamente, se presentan en la Figura 7.9 los resultados obtenidos en R:

```
> boxplot(TPC ~ Algoritmo, data=dat,main="ALGORITMO",ylab="TPC", plot=F)
$stats
     [,1] [,2]
[1,]  376  778
[2,]  684 1150
[3,] 1537 3400
[4,] 1593 4550
[5,] 1593 6520

$n
[1] 5 5

$conf
          [,1]      [,2]
[1,]  894.7029  997.5686
[2,] 2179.2971 5802.4314

$out
[1] 3170

$group
[1] 1

$names
[1] "1" "2"
```

Figura 7.9: Estadísticas en R del *boxplot* algoritmo

Seguidamente, se muestra en la Figura 7.10 el *boxplot* correspondiente a los escenarios.

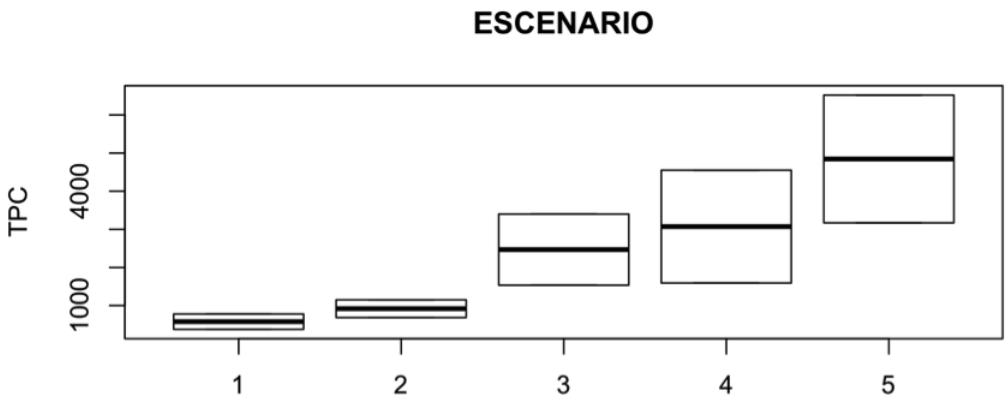


Figura 7.10: *Boxplot* por escenario

De la Figura 7.10 y tomando en cuenta los resultados de la herramienta estadística R, se puede decir de los escenarios que presentan un $n = 2$ correspondiente a los dos algoritmos planteados y explicados en capítulos previos lo siguiente:

1. Escenario-1 tuvo un límite inferior y un percentil 25 de 376 segundos, un percentil 50 o sea la mediana de 577 segundos y un percentil 75 con un valor máximo de 778 segundos.

2. Escenario-2, se tuvo un límite inferior y un percentil 25 de 684 segundos, un percentil 50 o sea la mediana de 917 segundos y un percentil 75 con un valor máximo de 1150 segundos.

3. Escenario-3, se tuvo un límite inferior y un percentil 25 de 1593 segundos, un percentil 50 o sea la mediana de 2468,5 segundos y un percentil 75 con un valor máximo de 3400 segundos.

4. Escenario-4, se tuvo un límite inferior y un percentil 25 de 1593 segundos, un percentil 50 o sea la mediana de 3071,5 segundos y un percentil 75 con un valor máximo de 4550 segundos.

5. Escenario-5 presentó un límite inferior y un percentil 25 de 3170 segundos, un percentil 50 o sea la mediana de 4845 segundos y un percentil 75 con un valor máximo de 6520 segundos.

Seguidamente, se presentan en la Figura 7.11 los resultados obtenidos en R:

```
> boxplot(TPC ~ Escenario, data=dat,main="ESCENARIO",ylab="TPC", plot=F)
$stats
     [,1] [,2]   [,3]   [,4] [,5]
[1,]  376  684 1537.0 1593.0 3170
[2,]  376  684 1537.0 1593.0 3170
[3,]  577  917 2468.5 3071.5 4845
[4,]  778 1150 3400.0 4550.0 6520
[5,]  778 1150 3400.0 4550.0 6520

$n
[1] 2 2 2 2 2

$conf
          [,1]      [,2]      [,3]      [,4]     [,5]
[1,]  127.8741  396.3714  387.1029 -232.1453 1102.284
[2,] 1026.1259 1437.6286 4549.8971 6375.1453 8587.716

$out
numeric(0)

$group
numeric(0)

$names
[1] "1" "2" "3" "4" "5"
```

Figura 7.11: Estadísticas en R del *boxplot* escenario

7.0.8. Análisis de desempeño del algoritmo propuesto

A continuación se presenta un análisis del desempeño del algoritmo, tomando en cuenta: tiempo de *commit*, transacciones en conflicto, transacciones sin conflicto y considerando los cinco escenarios estudiados. A su vez, se analiza la complejidad algorítmica de la nueva propuesta con el fin de dar explicación al comportamiento de los resultados obtenidos.

7.0.8.1. Tiempo de *commit* y tasas de transacciones

En relación con el tiempo de *commit* de la Figura 7.12, se puede deducir que se mantuvieron muy semejantes los resultados obtenidos por ambos móviles, donde se dio una diferencia para el escenario-1 con 200 transacciones de 50 segundos, mientras que para el escenario-2 con 400 transacciones de 31 segundos. Sucedió lo mismo para el caso del escenario-3 que con 800 transacciones hubo una diferencia de 25 segundos. De ahí que la poca diferencia mencionada se ve reflejada en la Figura 7.12, donde la gráfica presenta prácticamente la misma posición excepto por las diferencias ya indicadas. En este caso, lo interesante es destacar cómo a pesar de haber existido un incremento en las transacciones, el comportamiento del algoritmo se mantuvo similar para ambos dispositivos en cuanto a su tiempo de conclusión transaccional.
Por otra parte, en el caso de la Figura 7.13, la diferencia sí fue más notable pero sobre todo entre el dispositivo-2 y el dispositivo-3, no así entre el dispositivo-1 y el dispositivo-2. Entre el dispositivo-1 y el dispositivo-2, existió para el escenario-4 una diferencia de 26 segundos con 1200 transacciones, mientras que para el escenario-5 la diferencia fue de 11 segundos con 2400 transacciones. En el caso de la diferencia entre el dispositivo-2 y dispositivo-3 fue de 52 segundos para el escenario-4 y de 79 segundos para el escenario-5. De ahí que en la Figura 7.13 se pueda ver una diferencia más notable entre el dispositivo-3 con respecto al dispositivo-2 que la que se da entre el dispositivo-2 y el dispositivo-1 que presentan diferencia en segundos poco significativa.

En la Figura 7.14 correspondiente a las transacciones con conflicto, escenarios 1-3, es notable que la tasa porcentual estuvo entre 30.52 % y 36.12 % entre valores mínimos y máximos, manteniéndose una diferencia de aproximadamente 5.6 % en términos generales. Al igual que en el análisis anterior, la tasa de transacciones en conflicto se mantiene constante aún con el aumento transaccional existente.

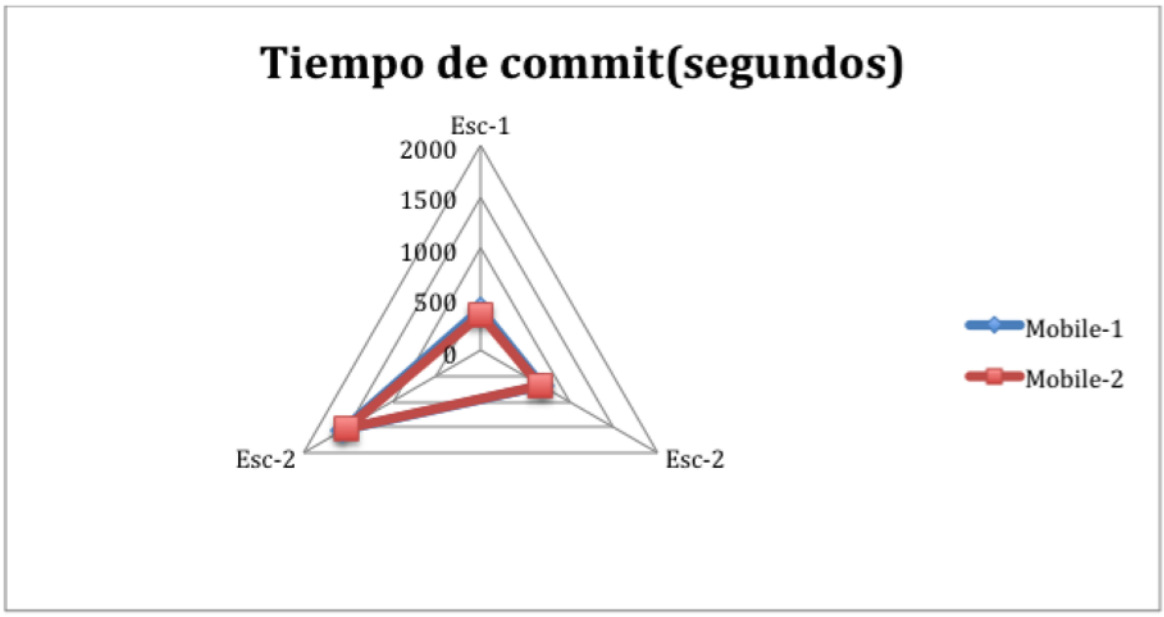


Figura 7.12: Tiempo de *commit*(segundos), escenarios 1-3

En la Figura 7.15 correspondiente a las transacciones con conflicto, escenarios 4-5, es notable que la tasa porcentual estuvo entre 40.81 % y 61.12 % entre valores mínimos y máximos, manteniéndose así una diferencia de aproximadamente 20.32 % en términos generales. En este caso, la diferencia porcentual es más significativa, siendo un importante aspecto por contemplar el aumento transaccional y de dispositivos en estudio.

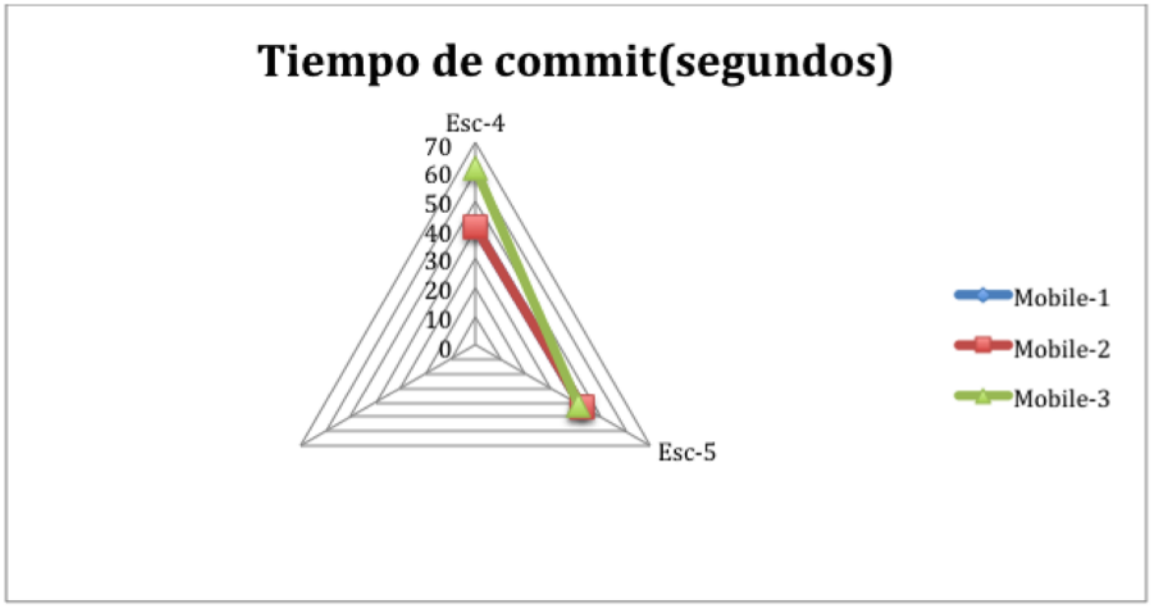


Figura 7.13: Tiempo de *commit*(segundos), escenarios 4-5

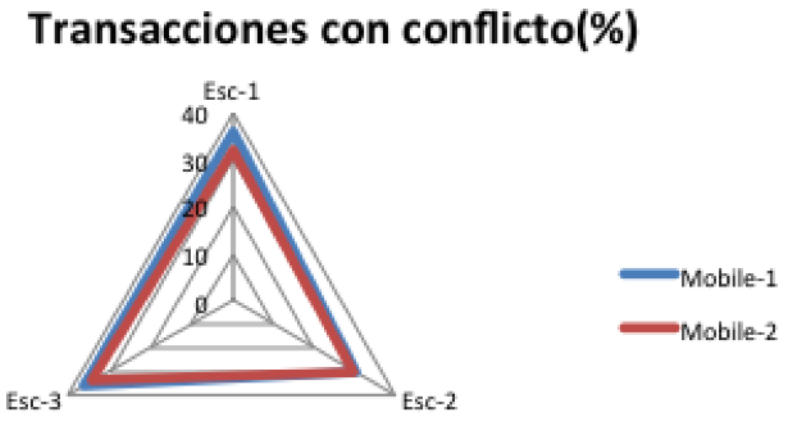


Figura 7.14: Transacciones con conflicto(%), escenarios 1-3

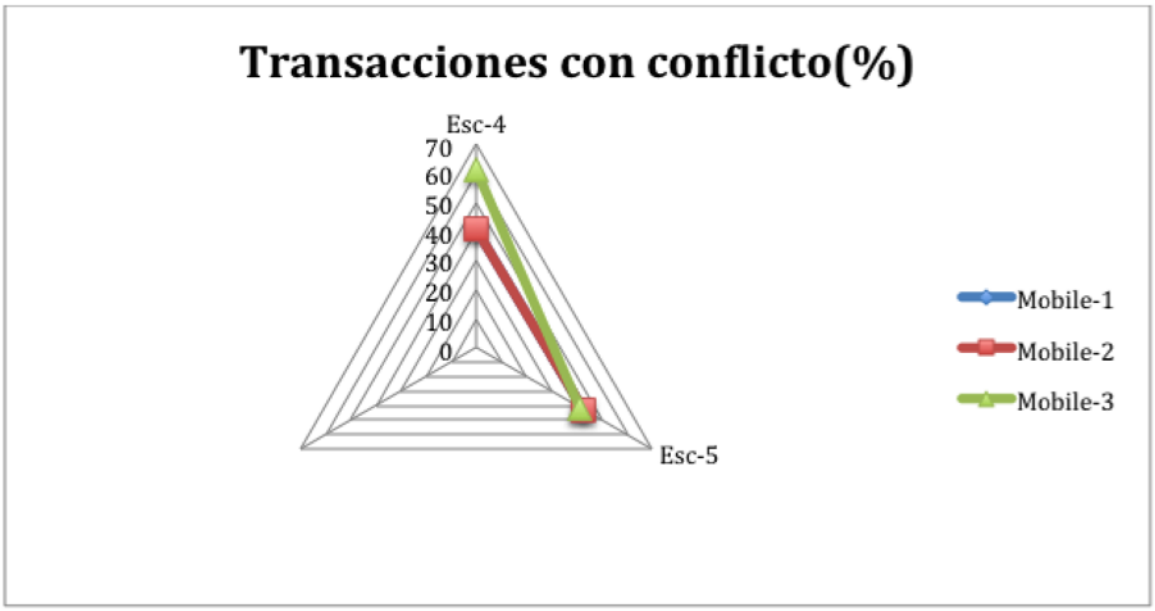


Figura 7.15: Transacciones con conflicto(%), escenarios 4-5

En la Figura 7.16 correspondientes a las transacciones sin conflicto, escenarios 1-3 es de señalar que la tasa porcentual estuvo entre 63.88 % y 70.18 % entre valores mínimos y máximos, manteniéndose una diferencia de aproximadamente 6.3 % en términos generales. Lo que quiere decir que al igual que en análisis anteriores, para estos escenarios el algoritmo se mantiene constante a pesar del aumento transaccional.

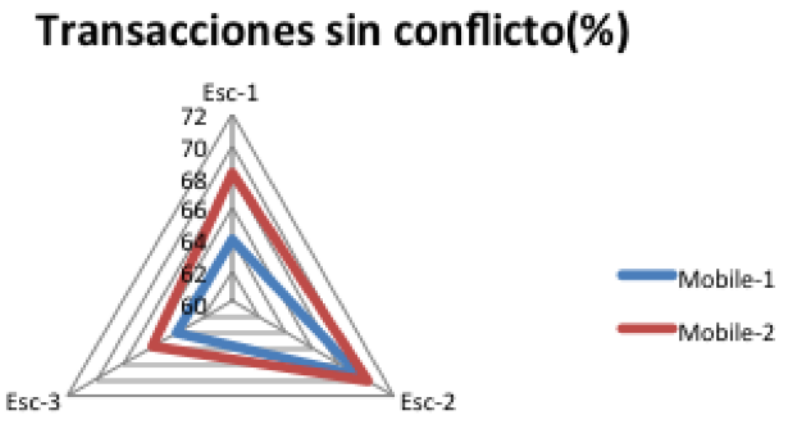


Figura 7.16: Transacciones sin conflicto(%), escenarios 1-3

En la Figura 7.17 correspondiente a las transacciones sin conflicto, escenarios 4-5, la tasa porcentual estuvo entre 38.88 % y 59.19 % entre valores mínimos y máximos, manteniéndose de este modo una diferencia de aproximadamente 20.31 % en términos generales. En este caso, sí hay un dato interesante de analizar como lo es la tasa con conflicto obtenida por parte del móvil-3 en el escenario-4, ya que el porcentaje fue de 38.88 %, el cual podemos considerar como un porcentaje muy bajo con respecto a los obtenidos y lo más destacable que en el escenario-5 la tasa se mantuvo en relación a las demás aún procesando el doble de las transacciones. Lo anterior se puede interpretar como un *outlier* o un valor que se encuentra fuera del comportamiento normal del algoritmo.

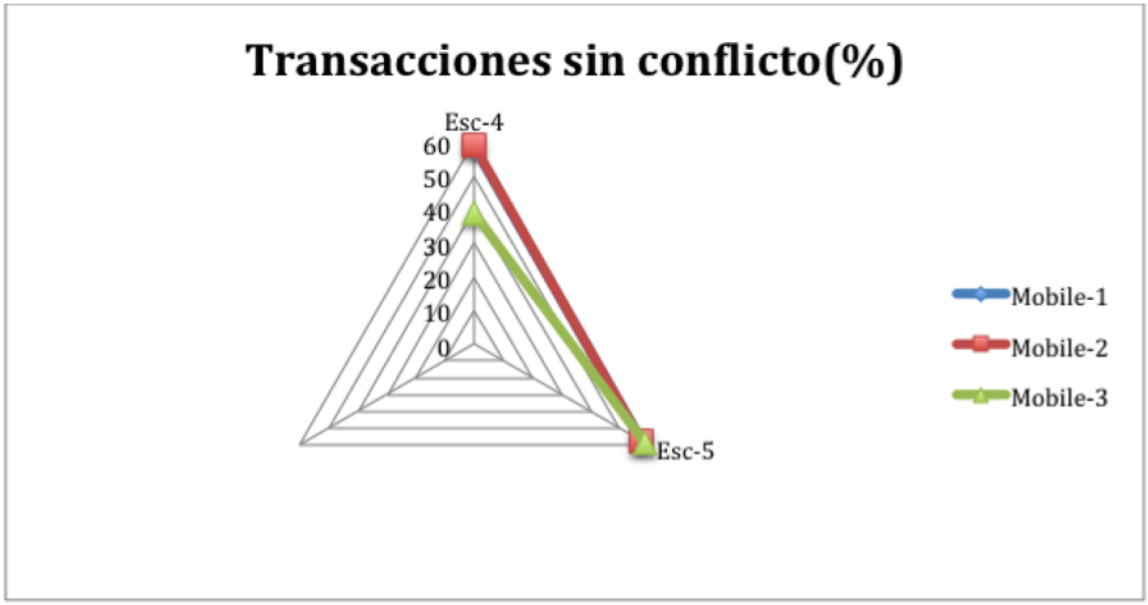


Figura 7.17: Transacciones con conflicto(%), escenarios 4-5

7.0.8.2. Análisis de complejidad algorítmica

Seguidamente se presenta el algoritmo del cliente con la complejidad algorítmica por línea de código implementada. Lo importante es que la complejidad algorítmica en el caso del cliente fue de $\theta(n)$ correspondiente al procedimiento de respaldo de las transacciones cuando ocurre una desconexión por parte del servidor o bien del dispositivo. Para el estudio de la complejidad algorítmica se tomaron como base los estudios de [67], [68] , [69], [70] y [71]

7.0.9. Algoritmo cliente (*Mobile Host*)

En el caso del análisis algorítmico del servidor, se tiene que presentará una complejidad algorítmica por línea de código implementada al igual que el análisis anterior y que en este caso se tienen dos escenarios:

1. Primer escenario. Si se tomara en consideración el ciclo infinito del servidor, el cual es parte de la implementación del modelo cliente-servidor se tendría una complejidad de $\theta(n)$ en este caso y de $\theta(n)$ en el caso de

```
1: Inicializate queueTrans, queueWait y queueDesc // θ(1)
2: CreateTransactions // θ(n)
3: if backupDB.getQueueDesconex()==null // θ(1) then
4:    Sin Transacciones desconectadas // θ(1)
5: else
6:    Transacciones desconectadas // θ(1)
7: end if
8: while !queueDesc AND !queueWait AND !queueTrans // θ(n) do
9:    read Transaction // θ(1)
10:   if connectFixed-Host // θ(1) then
11:      createSocket // θ(1)
12:      writeTransaction // θ(1)
13:      if TransacctionNotif // θ(1) then
14:         ExecuteTransaction // θ(1)
15:         print Successful Transaction // θ(1)
16:         Noconflict = Noconflict + 1 // θ(1)
17:      else
18:         NoExecuteTransaction // θ(1)
19:         print Conflict Transaction // θ(1)
20:         add QueueWait // θ(1)
21:         conflict = conflict + 1 // θ(1)
22:      end if
23:   else
24:      Exception
25:   end if
26:   sleep(RandomNumber (5000)) // θ(1)
27:   closeSocket // θ(1)
28:   if IOException then
29:      BackupDB // θ(1)
30:   end if
31:   if InterruptedException then
32:      BackupDB // θ(1)
33:   end if
34:   if ClassNotFoundException then
35:      BackupDB // θ(1)
36:   end if
37: end while
38: print Tiempo de Commit // θ(1)
39: print Transacciones con Conflicto // θ(1)
40: print Transacciones sin Conflicto // θ(1)
```

Algoritmo 3: Complejidad algorítmica Cliente (*Mobile Host*)

la búsqueda efectuada para encontrar transacciones en conflictos, por lo que nos daría una complejidad algorítmica de $\theta(n^2)$.

2. Segundo escenario. Sí se toma en cuenta que en realidad la aplicación de las transacciones de una estación móvil es de complejidad $\theta(1)$, ya que el $\theta(n)$ del ciclo infinito corresponde a todas las posibles conexiones que se puedan dar por parte de las estaciones móviles entonces la complejidad algorítmica sería de $\theta(n)$ correspondiente a la búsqueda únicamente.

Por lo tanto, para efectos de la presente investigación se analizará la complejidad basada en el segundo escenario y con lo cual se puede concluir que la complejidad algorítmica real de la propuesta es de $\theta(n)$.

7.0.10. Algoritmo servidor (*Fixed Host*)

```
1: Inicializate queueTrans
2: while TRUE do
3:    create Socket
4:    read TIn
5:    TIn.setTimeStamp(TRUE)
6:    TIn.setTimeStamp(ts)
7:    ts = ts + 1
8:    queueTrans.add(TIn)
9:    queueTransAux = queueTrans
10:   while queueTransAux AND queueTransAux.size > 1 do
11:      TOut = queueTransAux.poll
12:      if TIn.IdMH()=TOut.IdMH() AND
         TIn.DataItem()=TOut.DataItem() AND TOut.tipOp()=W then
13:         if TIn.tipOp()=R then
14:            TIn.setNotif(FALSE)
15:         end if
16:         if TIn.tipOp()=W then
17:            if TIn.TimeStamp() > tOut.TimeStamp() then
18:               TIn.setNotif(FALSE)
19:            end if
20:         end if
21:      end if
22:      if TIn.IdMH()=TOut.IdMH() AND
         TIn.DataItem()=TOut.DataItem() then
23:         queueTransAux.remove(TIn)
24:      end if
25:   end while
26: end while
```

Algoritmo 4: Servidor (*Mobile Host*)

```
1: Inicializate queueTrans // θ(1)
2: while TRUE // θ(n) do
3:    create Socket // θ(1)
4:    read TIn // θ(1)
5:    TIn.setTimeStamp(TRUE) // θ(1)
6:    TIn.setTimeStamp(ts) // θ(1)
7:    ts = ts + 1 // θ(1)
8:    queueTrans.add(TIn) // θ(1)
9:    queueTransAux = queueTrans // θ(1)
10:   while queueTransAux AND queueTransAux.size > 1 // θ(n) do
11:      TOut = queueTransAux.poll // θ(1)
12:      if TIn.IdMH()=TOut.IdMH() AND
         TIn.DataItem()=TOut.DataItem() AND TOut.tipOp()=W // θ(1)
         then
13:         if TIn.tipOp()=R // θ(1) then
14:            TIn.setNotif(FALSE) // θ(1)
15:         end if
16:         if TIn.tipOp()=W // θ(1) then
17:            if TIn.TimeStamp() > tOut.TimeStamp() then
18:               TIn.setNotif(FALSE) // θ(1)
19:            end if
20:         end if
21:      end if
22:      if TIn.IdMH()=TOut.IdMH() AND
         TIn.DataItem()=TOut.DataItem() // θ(1) then
23:         queueTransAux.remove(TIn) // θ(1)
24:      end if
25:   end while
26: end while
```

Algoritmo 5: Complejidad algorítmica Servidor (*Fixed Host*)

Capítulo 8

CONCLUSIONES GENERALES Y TRABAJO FUTURO

El planteamiento de un modelo en el contexto del control de la concurrencia con lleva a considerar varios elementos propios del ambiente móvil. Así por ejemplo, cuando se trabaja con dispositivos móviles es necesario tomar en cuenta la limitada capacidad y memoria que poseen este tipo de dispositivos. Además de considerar una serie de inconvenientes posibles dentro de los cuales están: el bajo ancho de banda, la latencia de la red y las desconexiones frecuentes.

A nivel de la presente investigación, el planteamiento de la pregunta de investigación sobre la caracterización de los modelos que resuelven el problema del control de la concurrencia, trajo consigo la incorporación a la nueva propuesta de una serie de características que permitieron no solo la posibilidad de garantizar su correcto desempeño, sino también lograr el objetivo planteado a nivel tanto de eficiencia como de eficacia.
Con el fin de contestar a la pregunta de investigación, es que se lista a continuación las principales características que tomando como referencia el modelo planteado debería tener un modelo móvil con desconexiones frecuentes:

- Implementar la aplicación utilizando el modelo cliente-servidor con el fin de poder utilizar diferentes tipos de escenarios distribuidos mediante un mismo gestor de base de datos, pero con varias instancias (clientes). Tanto el cliente como el servidor deberían estar representados con hilos (*threads*)

- Evitar bloqueos o cerrojos. Trabajar con un monitor implementado del lado del servidor que permita que se lleve a cabo cualquier tipo de transacción siempre y cuando no sea hacia un mismo esquema o tabla y que haya al menos una de esas transacciones que sea de escritura.
- No llevar a cabo abortos, es decir reiniciar recursos del sistema para atender las transacciones que presenten conflicto de concurrencia.
- Contemplar las desconexiones frecuentes. Trabajar almenos tres tipos de posibles desconexiones: por un problema de entrada y salida, por un problema de interrupción o bien por algún problema con la transacción entre el cliente y el servidor.
- Simular el tiempo que puede durar un usuario en llevar a cabo todo el proceso transaccional con el dispositivo móvil. Lo anterior se haría mediante una función aleatoria que calcule en milisegundos dicha actividad.
- Trabajar con estampillas de tiempo (*timestamp*) con el fin de evitar conflictos entre operaciones de escritura a un mismo esquema o tabla de una base de datos.
- Evitar los *delays*. Una alternativa es trabajar con abortos.
- No limitar el número de estaciones móviles que estén actualizándose.
- Procurar que el algoritmo pueda mantener constante su desempeño, aún cuando la cantidad de transacciones sea incremental.
- Procurar que el modelo sea híbrido es decir que no sea totalmente conservador.
- Permitir que el volumen transaccional sea alto , es decir hacer pruebas de hasta 2400 transacciones (800 transacciones por dispositivo).
- No incurrir en ciclos, permitir solución basada en casos, para resolver las situaciones de conflicto.

La lista anterior, permite dar una respuesta a la pregunta de investigación planteada en la presente investigación. Además con el fin de tener una base más solida de análisis, se llevó a cabo un estudio de Anova donde se logró

determinar las diferencias estadísticas tanto a nivel de escenarios como a nivel de algoritmos.

Cada uno de los objetivos fueron cumplidos de forma exitosa. Por ejemplo, en el objetivo relacionado con el análisis comparativo se hizo un análisis de 12 estudios con una distribución de 4 pesimistas, 4 optimistas y 4 conservadores. En este objetivo fue posible establecer las diferencias existentes entre las diferentes familias de algoritmos e identificar tantos las ventajas como las desventajas.

En el caso del objetivo relacionado con el planteamiento del modelo formal y algorítmico se pudo rescatar como novedoso el manejo de las desconexiones como un respaldo local en el dispositivo de las transacciones que no se terminaron de ejecutar. Otro de los elementos importantes, dentro de los modelos, es contemplar el tiempo de uso del dispositivo como si lo estuviera utilizando un usuario.

A nivel del objetivo que planteó implementar el modelo, lo rescatable fue utilizar tanto el cliente (*mobile host*) como el servidor (*fixed host*) a través del uso de *threads*, los cuales permitieron por las características de ser procesos ligeros o *lightweight* una programación que requirió de menos recursos computacionales. El otro aspecto importante de utilizar los *threads* fue el manejo de las desconexiones, ya que la propiedad de interrupción se facilitó por ser parte del uso de los *threads*.

En relación a la hipótesis de la investigación correspondiente a si existía o no diferencia significativa entre los diferentes escenarios, la cual correspondía al objetivo de evaluación del modelo, se rescató por medio del estudio de Anova y de diferencias significativas (Fisher LSD), que si existió diferencia significativa entre el algoritmo propuesto (JD-Algorithm) y el algoritmo estudiado (Moiz-Algorithm) específicamente entre el escenario 1 y el escenario-5, así como entre el escenario-2 y el escenario-5. Lo anterior no solo se puede explicar por la diferencia de transacciones las cuales según el escenario se van incrementando al doble, sino por el tiempo promedio de *commit* obtenido. Así por ejemplo para el caso del escenario-1, el tiempo promedio fue de 376 milisegundos y para el escenario-2 fue de 684,2 milisegundos. Contrario al escenario-5 que contó con 3170 milisegundos. Dándose una diferencia significativa clara.

En cuanto a la comparación entre algoritmos, el estudio estadístico dejó muy claro que la propuesta (JD-Algorithm) fue superior en cada uno de los es-

cenarios en relación con la propuesta (Moiz-Algorithm). Es importante resaltar que la propuesta (JD-Algorithm) no utilizó técnicas tradicionales de semáforos como sí lo hizo la propuesta (Moiz-Algorithm). Dentro de lo que sí compartieron fue el no reinicio de las transacciones y también el no manejo de los bloqueos.

A su vez en relación a la complejidad algorítmica se logró obtener un $\theta(n)$, que justifica el comportamiento del algoritmo de mantenerse en cuanto a los resultados de una manera constante.

Dentro de las limitaciones que presentó la nueva propuesta están:

- La técnica *Information Broadcasting* por su costo económico y complejidad de implementación, no fue tomada en cuenta.
- No se implementó prioridades en las transacciones excepto por el uso de los *timestamp* que representaron una prioridad de tiempo.
- El modelo no trabajó con transacciones anidadas, es decir que exista una transacción principal que tenga asociada otras sub-transacciones. En el modelo planteado, cada transacción es independiente entre sí, es decir tiene sus propios recursos computacionales aunque esto no significa que no pueda acceder un mismo esquema o tabla.
- No se contempla la propiedad de *timeout* ya que no fue necesario el uso de bloqueos.
- El uso de semáforos u otros mecanismos de espera no fueron empleados. Excepto el *sleep* en la implementación, como parte de la simulación del proceso de uso del dispositivo por parte de los usuarios.

Como se ha explicado el cumplimiento de los objetivos planteados fue hecha de una manera exitosa, aunque como en todo proceso de desarrollo de una investigación quedan aspectos que no se lograron cumplir y que pueden ser considerados como trabajo futuro. Es de esta forma que se podrá considerar como trabajo futuro:

- Manejar las transacciones no solo con *timestamp* tal y como se hizo sino con algún criterio de prioridad.
- Manejar transacciones anidadas con estructuras más complejas donde los recursos sean compartidos.

- Manejar la tecnología de agentes inteligentes para simular el paso de mensajes y el *broadcasting*.
- Incorporar en el análisis más estaciones de trabajo y si es posible heterogéneas, es decir que no sean de un mismo tipo de dispositivo tales como: *Palms*, *Pocket PC* y *Tablets*.
- Pasar de la simulación a la realidad utilizando la intervención del usuario dentro del proceso para validar mejor la manipulación del dispositivo.

Bibliografía

[1] J. B. Lim, A. R. Hurson, and K. M. Kavi, "Concurrent data access in mobile heterogeneous systems," in *Systems Sciences, 1999. HICSS-32. Proceedings of the 32nd Annual Hawaii International Conference on.* IEEE, 1999, pp. 10–pp.

[2] S. A. Moiz and L. Rajamani, "Concurrency control strategy to reduce frequent rollbacks in mobile environments," in *Computational Science and Engineering, 2009. CSE'09. International Conference on*, vol. 2. IEEE, 2009, pp. 709–714.

[3] S. A. Moiz and M. K. Nizamuddin, "Concurrency control without locking in mobile environments," in *Emerging Trends in Engineering and Technology, 2008. ICETET'08. First International Conference on.* IEEE, 2008, pp. 1336–1339.

[4] M. Wooldridge, *An introduction to multiagent systems.* John Wiley & Sons, 2009.

[5] A. Mhatre and R. Shedge, "Comparative study of concurrency control techniques in distributed databases," in *Communication Systems and Network Technologies (CSNT), 2014 Fourth International Conference on.* IEEE, 2014, pp. 378–382.

[6] N. Prabhu, V. Kumar, I. Ray, and G.-C. Yang, "Concurrency control in mobile database systems," in *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, vol. 2. IEEE Computer Society, 2004, pp. 83–83.

[7] W. Yendluri, A Hou and C. Wang, "Improving concurrency control in mobile databases," *Springer-Verlag Berlín Heidelberg DASFAA 2004, LNCS 2973*, pp. pp 642–655, 2004.

[8] J. B. Lim and A. R. Hurson, "Transaction processing in mobile, heterogeneous database systems," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 14, no. 6, pp. 1330–1346, 2002.

[9] O. A. Rawashdeh, H. A. Muhareb, and N. A. Al-Sayid, "An optimistic approach in distributed database concurrency control," in *Computer Science and Information Technology (CSIT), 2013 5th International Conference on*. IEEE, 2013, pp. 71–75.

[10] P. Serrano-Alvarado, C. Roncancio, M. Adiba, C. Labbé *et al.*, "Adaptable mobile transactions and environment awareness," *19èmes Journées Bases de Données Avancées (BDA)*, 2003.

[11] Y. Shirota, A. Iizawa, H. Mano, and T. Yano, "The echo method: concurrency control method for a large-scale distributed database," in *Data Engineering, 1999. Proceedings., 15th International Conference on*. IEEE, 1999, pp. 174–183.

[12] A. S. Al-Mogren and M. H. Dunham, "Buc, a simple yet efficient concurrency control technique for mobile data broadcast environment," pp. 564–569, 2001.

[13] D. Roldán Martínez, *Comunicaciones Inalámbricas, Un enfoque aplicado*. México, Alfaomega, Ra-Ma, 2005.

[14] F. M. de Assis Silva and S. Krause, "A distributed transaction model based on mobile agents," in *Mobile Agents*. Springer, 1997, pp. 198–209.

[15] A. Di Stefano, L. Lo Bello, and C. Santoro, "A distributed heterogeneous database system based on mobile agents," in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 1998.(WET ICE'98) Proceedings., Seventh IEEE International Workshops on*. IEEE, 1998, pp. 223–228.

[16] F. M. A. Silva and R. Popescu-Zeletin, "An approach for providing mobile agent fault tolerance," in *Mobile Agents*. Springer, 1998, pp. 14–25.

[17] D. B. Lange and M. Oshima, "Seven good reasons for mobile agents," *Commun. ACM*, vol. 42, no. 3, pp. 88–89, Mar. 1999. [Online]. Available: http://doi.acm.org/10.1145/295685.298136

[18] G. Liao, Y. Liu, L. Wang, and C. Peng, "Concurrency control of real-time transactions with disconnections in mobile computing environment," in *Computer Networks and Mobile Computing, 2003. ICCNMC 2003. 2003 International Conference on.* IEEE, 2003, pp. 205–212.

[19] X. WeiJun, L. Zhengding, L. Bing, and M. Sarem, "Transaction management in mobile multidatabase systems," in *Computer Networks and Mobile Computing, 2001. Proceedings. 2001 International Conference on*, 2001, pp. 513–518.

[20] J. Gray and A. Reuter, *Transaction processing.* Morgan Kaufíann Publishers, 1993.

[21] A. Das and K. Kai, "Tradeoff between client and server transaction validation in mobile environment," in *Database Engineering and Applications, 2001 International Symposium on.*, 2001, pp. 265–272.

[22] X. Song and J. W. S. Liu, "Maintaining temporal consistency: pessimistic vs. optimistic concurrency control," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 7, no. 5, pp. 786–796, Oct 1995.

[23] Y.-H. Loh, T. Hara, M. Tsukamoto, and S. Nishio, "A hybrid method for concurrent updates on disconnected databases in mobile computing environments," in *Proceedings of the 2000 ACM symposium on Applied computing-Volume 2.* ACM, 2000, pp. 563–565.

[24] A. Chianese, A. d'Acierno, V. Moscato, and A. Picariello, "Preserialization of long running transactions to improve concurrency in mobile environments," in *Data Engineering Workshop, 2008. ICDEW 2008. IEEE 24th International Conference on*, April 2008, pp. 129–136.

[25] S. A. Moiz and J. Supriya N.Pal, "Concurrency control in mobile environments: Issues and challenges," in *International Journal of Database Management Systems (IJDMS)*, 2011.

[26] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The notions of consistency and predicate locks in a database system," *Communications of the ACM*, vol. 19, no. 11, pp. 624–633, 1976.

[27] A. Adya, R. Gruber, B. Liskov, and U. Maheshwari, "Efficient optimistic concurrency control using loosely synchronized clocks," *SIGMOD Rec.*, vol. 24, no. 2, pp. 23–34, May 1995. [Online]. Available: http://doi.acm.org/10.1145/568271.223787

[28] Q. Mao, J. Wang, and Y. Zhan, "The optimistic locking concurrency controlling algorithm based on relative position and its application in real-time collaborative editing system," in *Computer Supported Cooperative Work in Design, 2004. Proceedings. The 8th International Conference on*, vol. 1, May 2004, pp. 99–105 Vol.1.

[29] Z. Itani, H. Diab, and H. Artail, "Optimistic pull based replication for mobile databases," in *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, vol. 2, June 2005, pp. 895–900 vol.2.

[30] T. Bai, Y. Liu, and Y. Hu, "Timestamp vector based optimistic concurrency control protocol for real-time databases," in *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, Oct 2008, pp. 1–4.

[31] F. Laux and M. Laiho, "Sql access patterns for optimistic concurrency control," in *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD '09. Computation World:*, Nov 2009, pp. 254–258.

[32] A. Makni and R. Bouaziz, "Performance evaluation of an optimistic concurrency control algorithm for temporal databases," in *Advances in Databases Knowledge and Data Applications (DBKDA), 2010 Second International Conference on*, April 2010, pp. 75–81.

[33] K.-Y. Lam, T.-W. Kuo, W.-H. Tsang, and G. C. Law, "Concurrency control in mobile distributed real-time database systems," *Information Systems*, vol. 25, no. 4, pp. 261–286, 2000.

[34] V. Kumar, N. Prabhu, M. H. Dunham, and A. Y. Seydim, "Tcot-a timeout-based mobile transaction commitment protocol," *Computers, IEEE Transactions on*, vol. 51, no. 10, pp. 1212–1218, 2002.

[35] P. K. Reddy and M. Kitsuregawa, "Speculative lock management to increase concurrency in mobile environments," in *Mobile Data Access*. Springer, 1999, pp. 82–96.

[36] M. K. Nizamuddin and S. A. Sattar, "Data count driven concurrency control scheme with performance gain in mobile environments," *J. Emerg. Trend. Comput. Inform. Sci*, vol. 2, no. 2, pp. 106–112, 2010.

[37] M. K. Nizamuddin and D. S. A. Sattar, "An improved, prioritized concurrency control scheme with performance gain in mobile environments," *ARPN Journal of Systems and Software*, vol. 11, 2010.

[38] Z. Xing, L. Gruenwald, and K. Phang, "Soda: An algorithm to guarantee correctness of concurrent transaction execution in mobile p2p databases," in *Database and Expert Systems Application, 2008. DEXA'08. 19th International Workshop on*. IEEE, 2008, pp. 337–341.

[39] Z. Koo and S. Moon, "Effects of broadcast errors on concurrency control in wireless broadcasting environments," *Information processing letters*, vol. 86, no. 1, pp. 13–21, 2003.

[40] H.-T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," *ACM Transactions on Database Systems (TODS)*, vol. 6, no. 2, pp. 213–226, 1981.

[41] T. Härder, "Observations on optimistic concurrency control schemes," *Information Systems*, vol. 9, no. 2, pp. 111–120, 1984.

[42] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. Addison-wesley New York, 1987, vol. 370.

[43] M. K. Nizamuddin and S. A. Sattar, "Algorithm for priority based concurrency control without locking in mobile environments," in *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, vol. 3. IEEE, 2011, pp. 108–112.

[44] D. C. Montgomery, *Design and analysis of experiments*. John Wiley & Sons, 2008.

[45] H.-J. Choi and B.-S. Jeong, "A timestamp-based optimistic concurrency control for handling mobile transactions," in *Computational Science and Its Applications-ICCSA 2006*. Springer, 2006, pp. 796–805.

[46] J.-H. Yuen, E. Chan, K.-Y. Lam, and H.-W. Leung, "An adaptive avi-based cache invalidation scheme for mobile computing systems," in *Database and Expert Systems Applications, 2000. Proceedings. 11th International Workshop on*. IEEE, 2000, pp. 155–159.

[47] M. Lee and S. Helal, "Hicomo: High commit mobile transactions," *Distributed and Parallel Databases*, vol. 11, no. 1, pp. 73–92, 2002.

[48] N. Nouali, A. Doucet, and H. Drias, "A two-phase commit protocol for mobile wireless environment," in *Proceedings of the 16th Australasian database conference-Volume 39*. Australian Computer Society, Inc., 2005, pp. 135–143.

[49] P. Serrano-Alvarado, C. Roncancio, and M. Adiba, "A survey of mobile transactions," *Distributed and Parallel databases*, vol. 16, no. 2, pp. 193–230, 2004.

[50] S. A. Moiz and D. L. Rajamani, "An algorithmic approach for achieving concurrency in mobile environment," in *1st National Conference on Computing for Nation Development, INDIACom*, 2007.

[51] A. R. Pedersen and N. Ringgade, "Design and analysis of experiments," 1985.

[52] O. Kempthorne, "The design and analysis of experiments." 1952.

[53] R. E. Bechhofer, D. M. Goldsman, and T. J. Santner, *Design and analysis of experiments for statistical selection, screening, and multiple comparisons*. Wiley, 1995.

[54] T. P. Ryan, "Design and analysis of experiments," *Modern Engineering Statistics: Solutions Manual to Accompany*, pp. 139–153, 2000.

[55] S. S. Shapiro, *How to test normality and other distributional assumptions*. American Society for Quality Control, 1990, vol. 3.

[56] C. M. Jarque and A. K. Bera, "A test for normality of observations and regression residuals," *International Statistical Review/Revue Internationale de Statistique*, pp. 163–172, 1987.

[57] P. Royston, "Approximating the shapiro-wilk w-test for non-normality," *Statistics and Computing*, vol. 2, no. 3, pp. 117–119, 1992.

[58] N. M. Razali and Y. B. Wah, "Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests," *Journal of Statistical Modeling and Analytics*, vol. 2, no. 1, pp. 21–33, 2011.

[59] H. Hartley, "Testing the homogeneity of a set of variances," *Biometrika*, pp. 249–255, 1940.

[60] M. S. Bartlett and D. Kendall, "The statistical analysis of variance-heterogeneity and the logarithmic transformation," *Supplement to the Journal of the Royal Statistical Society*, pp. 128–138, 1946.

[61] M. B. Brown and A. B. Forsythe, "Robust tests for the equality of variances," *Journal of the American Statistical Association*, vol. 69, no. 346, pp. 364–367, 1974.

[62] M. E. O'Neill and K. L. Mathews, "Levene tests of homogeneity of variance for general block and treatment designs," *Biometrics*, vol. 58, no. 1, pp. 216–224, 2002.

[63] T.-S. Lim and W.-Y. Loh, "A comparison of tests of equality of variances," *Computational Statistics & Data Analysis*, vol. 22, no. 3, pp. 287–301, 1996.

[64] G. V. Glass, "Testing homogeneity of variances," *American Educational Research Journal*, vol. 3, no. 3, pp. 187–190, 1966.

[65] A. K. Bera, C. M. Jarque, and L.-F. Lee, "Testing the normality assumption in limited dependent variable models," *International Economic Review*, pp. 563–578, 1984.

[66] R. A. Fisher, "On the mathematical foundations of theoretical statistics," *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, pp. 309–368, 1922.

[67] E. Horowitz and S. Sahni, *Fundamentals of computer algorithms*. Computer Science Press, 1978.

[68] M. A. Weiss, *Data Structures and Algorithms*. Benjamin/Cummings, 1992.

[69] B. Gilles and B. Paul, "Fundamentals of algorithms," 1996.

[70] G. Brassard and P. Bratley, *Fundamentals of algorithmics*. Prentice-Hall, Inc., 1996.

[71] S. Dasgupta, C. H. Papadimitriou, and U. Vazirani, *Algorithms*. McGraw-Hill, Inc., 2006.

ANEXOS

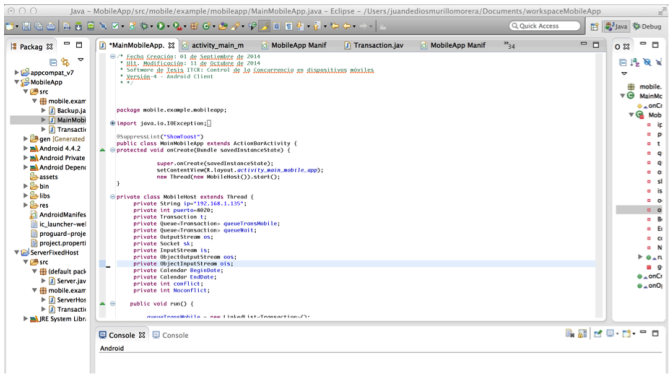


Figura 8.1: Cliente implementado en Java

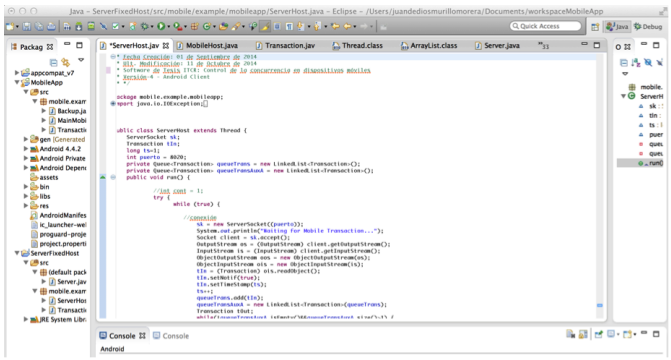


Figura 8.2: Servidor implementado en Java

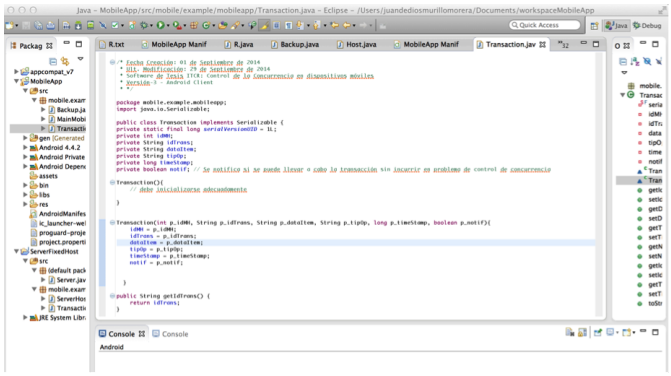
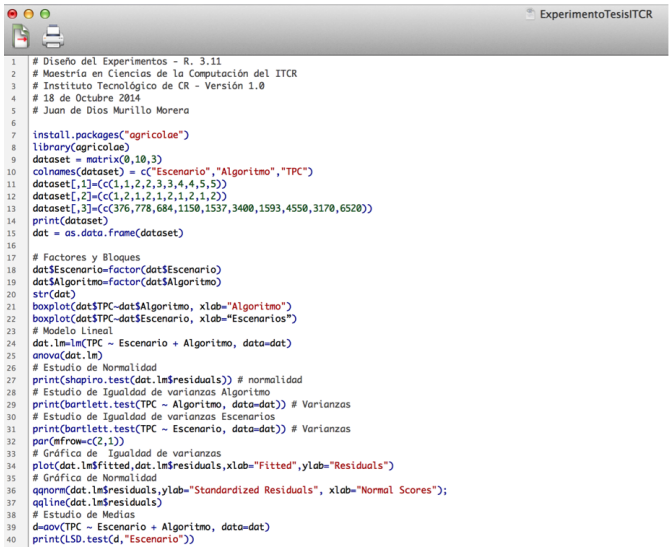


Figura 8.3: Objeto transacción implementado en Java



```
# Diseño del Experimentos - R. 3.11
# Maestria en Ciencias de la Computación del ITCR
# Instituto Tecnológico de CR - Versión 1.0
# 18 de Octubre 2014
# Juan de Dios Murillo Morera

install.packages("agricolae")
library(agricolae)
dataset = matrix(0,10,3)
colnames(dataset) = c("Escenario","Algoritmo","TPC")
dataset[,1]=c(1,1,2,2,3,3,4,4,5,5)
dataset[,2]=c(1,2,1,2,1,2,1,2,1,2)
dataset[,3]=c(376,778,684,1150,1537,3400,1593,4550,3170,6520)
print(dataset)
dat = as.data.frame(dataset)

# Factores y Bloques
dat$Escenario=factor(dat$Escenario)
dat$Algoritmo=factor(dat$Algoritmo)
str(dat)
boxplot(dat$TPC~dat$Algoritmo, xlab="Algoritmo")
boxplot(dat$TPC~dat$Escenario, xlab="Escenarios")
# Modelo Lineal
dat.lm=lm(TPC ~ Escenario + Algoritmo, data=dat)
anova(dat.lm)
# Estudio de Normalidad
print(shapiro.test(dat.lm$residuals)) # normalidad
# Estudio de Igualdad de varianzas Algoritmo
print(bartlett.test(TPC ~ Algoritmo, data=dat)) # Varianzas
# Estudio de Igualdad de varianzas Escenarios
print(bartlett.test(TPC ~ Escenario, data=dat)) # Varianzas
par(mfrow=c(2,1))
# Gráfica de  Igualdad de varianzas
plot(dat.lm$fitted,dat.lm$residuals,xlab="Fitted",ylab="Residuals")
# Gráfica de Normalidad
qqnorm(dat.lm$residuals,ylab="Standardized Residuals", xlab="Normal Scores");
qqline(dat.lm$residuals)
# Estudio de Medias
d=aov(TPC ~ Escenario + Algoritmo, data=dat)
print(LSD.test(d,"Escenario"))
```

Figura 8.4: Experimento implementado en R

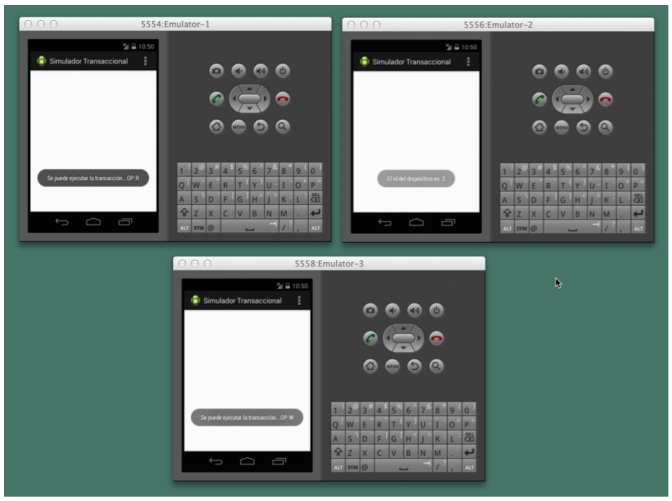


Figura 8.5: Transacciones sin conflicto

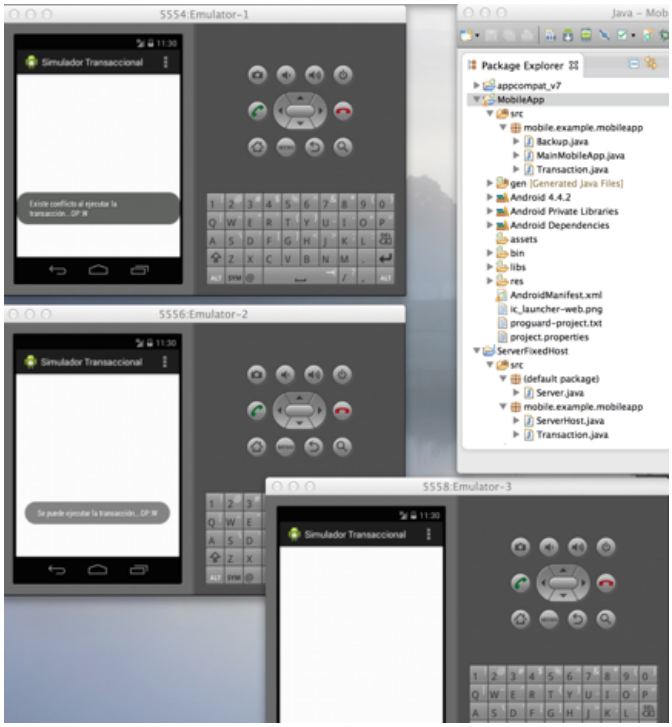


Figura 8.6: Transacciones con conflicto

Figura 8.7: Reporte de tiempo de commit

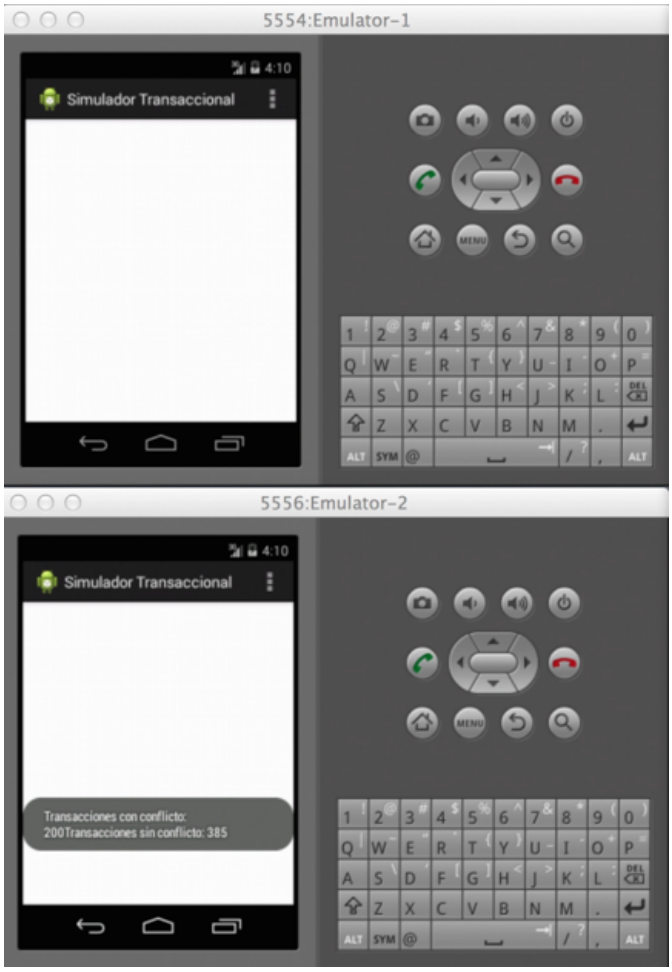


Figura 8.8: Reporte de transacciones con conflicto y sin conflicto

Figura 8.9: Tipos de desconexiones en Android

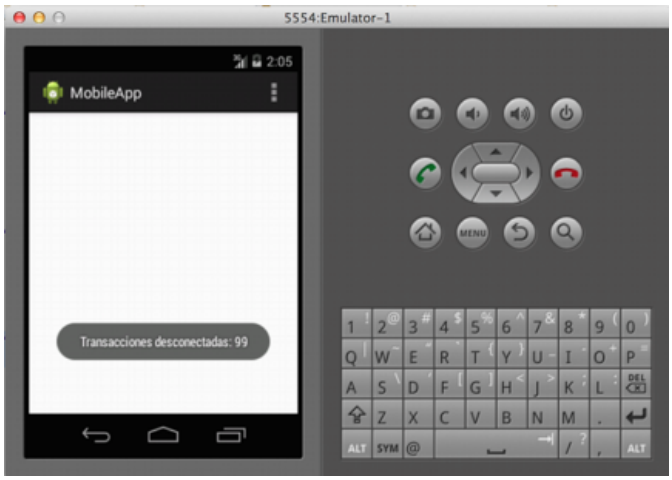


Figura 8.10: Transacciones recuperadas después de una caída

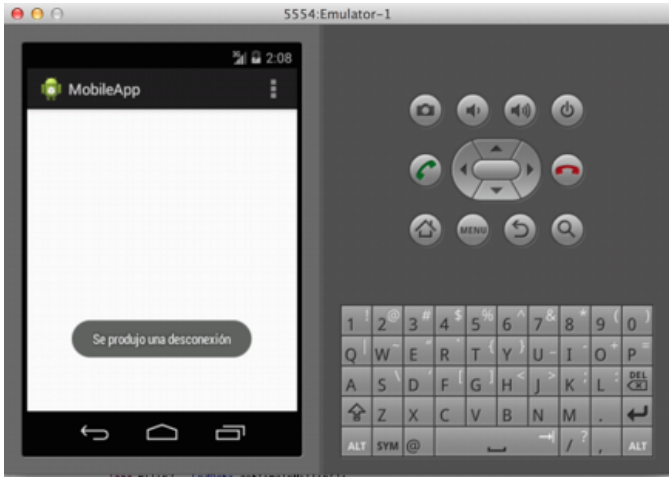


Figura 8.11: Aviso de desconexión