UNIVERSIDAD
DE LOS ANDES
VENEZUELA

Proyecto de Grado

Presentado ante la ilustre Universidad de Los Andes como requisito parcial para obtener el Título de Ingeniero de Sistemas

Desarrollo de un sistema de procesamiento de tráfico multihilo para la compañía Ogangi de Venezuela C.A.

Por

Br. Katherine Andrade

Tutor: Prof. Gerard Páez

Asesor: Ing. Grisel D'Alessio

Febrero 2020

Desarrollo de un sistema de procesamiento de tráfico multihilo para la compañía Ogangi de Venezuela C.A.

Br. Katherine Andrade

Proyecto de Grado — Sistemas Computacionales, 69 páginas
Escuela de Ingeniería de Sistemas, Universidad de Los Andes, 2020

Resumen: La programación paralela utilizada en el desarrollo de aplicaciones ha permitido usar de forma eficiente los recursos del hardware del computador y mejorar el desempeño del software, este tipo de programación ha incrementado la velocidad del cálculo usando múltiples procesadores juntos resolviendo los problemas de congestión causados por programas que se ejecutan de manera secuencial. La programación paralela se puede implementar por medio de hilos, ya que en computadoras con varias CPUs, los hilos dividen los problemas en partes para que puedan ser ejecutados en diferentes procesos al mismo tiempo. El lenguaje de programación Java es un lenguaje útil para la programación paralela porque permite utilizar de manera directa los sistemas multinúcleos, es portable y multiplataforma; además cuenta con herramientas que permiten manejar de manera fácil los hilos del sistema. Actualmente la compañía Ogangi de Venezuela es una empresa que presta un servicio de mensajería corporativa, los clientes a los que se les ofrece este servicio buscan poder acceder a la información de una manera fácil. La forma de lograr el seguimiento de la información es llevando un control de todos los datos que circulan a través del servicio de la empresa (procesamiento de tráfico de datos), estos datos se guardan en archivos; cada línea de los archivos con los datos procesados tiene los detalle de un mensaje y recibe el nombre de CDR (Call Detail Record, por sus siglas en inglés). El procesamiento de tráfico de datos se hace con un sistema propio llamado CDRMaganger y se hace de forma secuencial causando un retraso en el procesamiento de los datos, esto trae como consecuencia problemas de congestión en la generación de reportes realizada por los clientes de la compañía, en la mayoría de los casos la información presenta retraso de varias horas. Por esta razón en este Proyecto de Grado, se desarrolló un sistema

multihilo para el procesamiento de tráfico de la compañía Ogangi de Venezuela C.A. con el objetivo de mejorar el tiempo de procesamiento de los registros de detalles de mensajes (CDRs) y un sistema de generación de reportes basado en tecnologías web para mejorar el acceso a la información actualizada por parte de los clientes.

Palabras clave: Programación paralea, Java, CDR, multihilo.

Este trabajo fue procesado en LaTeX.

Índice general

Índice de figuras

Índice de cuadros

Introducción

En los últimos años, ha habido un creciente interés en el desarrollo de programas utilizando programación paralela debido a que los requerimientos a nivel de procesamiento cada día aumentan. Así mismo, la mayor parte del hardware de las computadoras actuales, está hecho para que las tareas se realicen de manera paralela. La utilización de este tipo de programación para el desarrollo de aplicaciones permite mejorar el desempeño del software de manera general y adicionalmente se utilizan de una forma eficiente los recursos de hardware del computador.

De esta manera se pueden resolver los problemas de congestión causados por programas que se ejecutan de forma secuencial, estos problemas ocasionan que el desempeño de las computadoras se vea afectado. En los sistemas desarrollados utilizando técnicas de programación paralela, los procesos que normalmente padecen un embotellamiento, no afectan al desempeño del sistema ya que gran parte de los procesos del mismo se realizan de manera paralela.

Actualmente el procesamiento de tráfico de datos del programa CDRManager de la compañía Ogangi de Venezuela C.A. se hace de manera secuencial, lo que causa un retraso en el procesamiento de los datos y por ende problemas a los clientes a la hora de generar los reportes ya que la información no está actualizada. Por esta razón se implementó un sistema multihilo para el procesamiento de tráfico, cuyo objetivo es mejorar el tiempo de procesamiento de los registros de detalles de mensajes. Adicionalmente se creó un sistema web que le facilite a los clientes poder generar los reportes que necesiten.

Capítulo 1

Contextualización

Ogangi de Venezuela C.A. es una empresa que presta un servicio de mensajería masiva y automatizada con la misión de poder acelerar y facilitar la evolución de servicios móviles de datos. Permitiendo a los operadores celulares incrementar más rápidamente el uso de sus servicios; a los proveedores y dueños de contenidos, crear una presencia móvil y a los usuarios móviles, disfrutar de los servicios a través de una experiencia satisfactoria. Esta empresa entrega millones de transacciones a través de las redes de datos móviles de América a través de su plataforma y actualmente procesa más de un millón de mensajes diarios, con sistemas informáticos redundantes, tolerantes a fallas y con soporte las 24 horas del día, por 7 días a la semana, los 365 días del año.

En la actualidad, el proyecto de negocio de la empresa Ogangi se divide principalmente en dos áreas:

- Mensajería corporativa.
- Desarrollo de aplicaciones móviles.

La mensajería corporativa, es un sistema corporativo de comunicación y trabajo colaborativo, que se integra en los procesos de negocio permitiendo agilizarlos, y manejar la información de manera eficiente en base a los requisitos de privacidad, control y seguridad que determine la empresa, y lo más importante, aumenta la productividad y reduce los costos.

Los clientes a los que se les ofrece este servicio buscan poder acceder a la información de una manera fácil. Para lograrlo, se hace un procesamiento del tráfico de datos, lo que esto quiere decir, es que se lleva un control de todos los datos que pasan por el servicio de la empresa, en este caso, de mensajes, los cuales se guardan en archivos. Cada línea de este archivo se llama CDR (Call Detail Record), esta línea tiene el detalle de un mensaje SMS.

En este capítulo se presentan los antecedentes de la investigación, los cuales comprenden investigaciones sobre la programación paralela y las aplicaciones actuales de este tipo de programación, estas investigaciones involucran métodos eficientes para el aprovechamiento de los núcleos de los microprocesadores. Luego, se hace mención a la utilización del lenguaje de programación Java para el desarrollo de programas en paralelo ya que este tipo de programación permite dividir los problemas para ser ejecutados en diferentes procesadores por medio de hilos de ejecución. También, se presenta el planteamiento del problema, la justificación, los objetivos generales y específicos, la metodología y el alcance de este proyecto de grado.

1.1. Antecedentes

En los últimos años, la evolución tecnológica ha llevado a desarrollar nuevos mecanismos para poder seguir con el desarrollo del aumento de la capacidad de procesamiento, duplicando o multiplicando los procesadores internos que componen un mismo circuito integrado [17]. Esto hace que haya una necesidad de buscar una manera de programación que permita poder utilizar completamente la potencialidad del cómputo integrado. La manera para lograr esto, es por medio de la programación paralela, ya que es utilizada para aumentar la velocidad del cálculo usando múltiples procesadores juntos [8]. Este tipo de programación se puede implementar por medio de hilos. Un proceso tiene un espacio de direcciones con un solo hilo de control, sin embargo, también se pueden tener varios hilos de control compartiendo el mismo espacio de direcciones y todos sus datos entre ellos. En los sistemas con varias CPUs, los hilos son útiles ya que hacen posible el verdadero paralelismo [15] dividiendo los problemas en partes para que puedan ser ejecutados en diferentes procesos.

Java es un lenguaje que permite aprovechar de forma directa los sistemas multi núcleos, es portable y multiplataforma. Lo que lo hace un lenguaje de programación de interés para la programación paralela [9, 13]. Este lenguaje también cuenta con clases las cuales permiten manejar de manera fácil los hilos del sistema [3], permitiendo utilizar este lenguaje para el desarrollo de programas en paralelo.

El lenguaje de programación Java ha sido utilizado ampliamente en investigaciones recientes para el desarrollo de aplicaciones con requerimientos de alto desempeño. A continuación se presentan algunas de estas investigaciones:

La investigación [11], titulada "A Scalable Architecture for Multi-threaded JAVA Applications", se presenta una arquitectura escalable para aplicaciones Java multihilo llamada JASIP. La arquitectura propuesta consiste en múltiples elementos de procesamientos específicos de la aplicación, cada uno capaz de ejecutar un solo hilo a la vez. La arquitectura se evalúa mediante la implementación de una máquina Java portátil y escalable en una placa FPGA para su demostración. Esta arquitectura combina las ventajas de Java con su mecanismo simple para manejar hilos y procesadores específicos de aplicaciones (ASIP) con su arquitectura altamente optimizada. La arquitectura paralela y escalable de JASIP refleja directamente la ejecución de la fuente Java utilizando hilos de manera concurrente. Por lo tanto, proporciona una arquitectura que está optimizada y adicionalmente registra altas aceleraciones debido a la ejecución paralela de los hilos.

En la investigación [16] titulada "Cifrado de datos con algoritmo AES usando programación multihilo en Java", presenta una implementación en software de un esquema de paralelismo a nivel de datos para el algoritmo de cifrado AES con una llave de 128 bits, en particular para computadoras de escritorio y portátiles con procesadores multi núcleo. Se hizo una implementación de la versión paralela del algoritmo AES, en lenguaje Java. Primero se divide el archivo de datos en un número de partes igual al número de núcleos del procesador o a un número de hilos deseado. El tamaño de cada parte del archivo se escoge de tal manera que sea múltiplo de 16 debido a que el algoritmo AES cifra en bloques de 128 bits o 16 bytes. En caso de que el tamaño de la última parte no sea múltiplo de 16 se completa con caracteres de relleno en un proceso llamado padding. Cada parte del archivo es asignada a un hilo de ejecución el cual

ejecuta el proceso de cifrado. El archivo se ensambla con las partes cifradas por cada hilo de ejecución, de acuerdo al orden en el que esa parte aparece en el archivo original. El proceso de descifrado es exactamente igual, la única diferencia consiste en que el hilo de ejecución descifra en lugar de cifrar. Se probó el desempeño en una computadora con un procesador Intel de 4 núcleos bajo un sistema operativo Linux. Las pruebas realizadas mostraron que se obtiene una disminución en el tiempo de cifrado de hasta 36 % a partir de 5 hilos de ejecución para tamaños de archivo mayores a 5 MB. Un número de hilos mayor a 5 ya no mejora significativamente el desempeño e incluso lo puede empeorar por la sobrecarga que impone al proceso y al sistema operativo manejar un número de hilos grande.

En la investigación [14], titulada "Design of Efficient Java Communications for High Performance Computing", existe un creciente interés en adoptar Java como el lenguaje de programación paralelo para la era de múltiples núcleos. Este interés exige un desempeño escalable en arquitecturas híbridas de memoria compartida/distribuida. Si bien Java ofrece ventajas importantes, como el multihilo integrado y el soporte de red, la seguridad, el uso generalizado, la alta productividad de programación, la portabilidad y la independencia de la plataforma, la falta de un middleware de comunicación eficiente es un inconveniente importante para su utilización en computación de alto desempeño (HPC, por sus siglas en inglés). Para esto, se ha diseñado, desarrollado y evaluado una implementación de sockets de Java de alto desempeño (llamada JFS, Java Fast Sockets), con el fin de aprovechar las redes de alta velocidad (SCI, Myrinet, InfiniBand) y la memoria compartida. Además, se proporciona un eficiente soporte de comunicación sin bloqueo a través del desarrollo de la biblioteca iodev, que permite superponer la comunicación y el cálculo. Finalmente, se implementó una biblioteca de paso de mensajes de Java denominada Fast MPJ (F-MPJ), que proporciona algoritmos más escalables en primitivas de comunicación colectiva. La conclusión final y principal es que el uso de Java para HPC es factible, e incluso recomendable cuando se busca un desarrollo productivo, siempre que se disponga de un middleware de comunicación eficiente, como los proyectos presentados en esta investigación, y siguiendo las pautas para la optimización del desempeño.

En [9], titulada "Java Thread and Process Performance for Parallel Machine

Learning on Multicore HPC Clusters", se habla sobre la importancia de los problemas de desempeño y el valor de la tecnología de computación de alto desempeño (HPC) en el uso creciente de los framework de Big Data en máquinas con una gran capacidad de procesamiento. Este documento analiza con atención tres frameworks principales como Spark, Flink y MPI (Interfaz de pase de mensajes, por sus siglas en inglés), tanto en la escala a través de nodos como internamente en los muchos núcleos dentro de los nodos modernos. Se centran en los desafíos especiales de la Máquina Virtual de Java (JVM) utilizando un clúster HP Haswell HPC con 24 núcleos por nodo. En este estudio de desempeño se utilizan dos algoritmos paralelos de aprendizaje automático, el agrupamiento K-Means y el escalamiento multidimensional (MDS, por sus siglas en inglés). Se identifican tres problemas principales: modelos de hilos, patrones de afinidad y mecanismos de comunicación, como factores que afectan el desempeño y se muestra cómo optimizarlos para que Java pueda igualar el desempeño de lenguajes HPC tradicionales como C. Las implementaciones actuales de los frameworks de computación de Big Data carecen de algoritmos de comunicación eficientes como los implementados en MPI. Se ha identificado la comunicación ineficiente como la característica más perjudicial para obtener el mejor rendimiento posible con Spark y Flink. Las deficiencias en el rendimiento que se observaron antes de las mejoras en las aplicaciones de aprendizaje automático de Java MPI se pueden ver en las implementaciones actuales de los tiempos de ejecución de Big Data como Flink y Spark, y se está trabajando para llevar estas mejoras a dichos framework. En particular, el objetivo es mejorar las comunicaciones colectivas de Flink y Spark utilizando algoritmos eficientes.

En el trabajo de investigación [12], titulado "Distributed High Performance Computing using JAVA", se realiza un estudio sobre el compromiso de usar Java para crear un clúster capaz de lograr computación de alto desempeño. Aquí se hace una arquitectura de computación distribuida en Java usando implementaciones de API de bajo nivel. La creación de una base de clústeres a través del método tradicional utilizando los lenguajes de programación C, C ++ o FORTRAN puede ser tediosa, compleja y requiere mucho tiempo. El clúster que se diseñó para lograr computación de alto desempeño requiere de una ejecución paralela para una computación más rápida. La arquitectura diseñada está especialmente enfocada en el modelo de programación

de datos múltiples de instrucción única (SIMD). Un solo problema se divide en tareas independientes más pequeñas para ser ejecutadas en paralelo. Para la aceleración de hardware, las GPU también se pueden agregar al clúster para el procesamiento vectorial de los datos para aumentar el cálculo del clúster implementando un paralelismo masivo de datos. Se utilizó una biblioteca de precisión arbitraria de alto desempeño para lograr una mayor precisión en los cálculos. La computación distribuida de alto desempeño utilizando Java es una visión para lograr un alto desempeño de computación utilizando API de bajo nivel y una estrategia de desarrollo de software básico utilizando Java.

Este documento muestra los beneficios e inconvenientes de usar Java para el desarrollo de aplicaciones distribuidas de alto desempeño. Se concluye que el uso de API de bajo nivel para crear HPC puede ser complejo en el caso de la programación de trabajos y la administración automática de recursos. Cuando se presenta en HPC distribuido, Java podría superar a otros lenguajes en caso de un nivel más alto de abstracción. Puede ser una buena opción para desarrollar computación distribuida debido a sus características como portabilidad, GUI (interfaz de usuario de gráficos), multihilo integrado y biblioteca de red. La heterogeneidad se puede tratar fácilmente con Java. Usar Java o no para HPC es la elección del usuario. Si el requisito incluye portabilidad, alta disponibilidad, desarrollo de software más sencillo, GUI o seguridad, Java será la mejor opción para ese tipo de aplicaciones.

1.2. Planteamiento del problema

La mensajería corporativa ha sido el negocio principal de Ogangi desde sus inicios. En la actualidad, esta empresa cuenta con una plataforma de mensajería ofreciendo a sus clientes cada vez más canales para la entrega de información (SMS, correo electrónico, notificaciones push, entre otros). Ogangi cuenta con una solución de procesamiento de tráfico y generación de reportes llamada CDRManager la cual procesa cada cierto tiempo todo el tráfico generado por los servicios de la empresa en los diferentes canales, los almacena en una estructura data warehouse[1] y los mantiene

[1]Un almacén de datos (del inglés data warehouse) es una colección de datos orientada a un determinado ámbito (empresa, organización, etc.), integrado, no volátil y variable en el tiempo, que ayuda a la toma de decisiones en la entidad en la que se utiliza.

actualizados y disponibles para ser consultados.

También cuenta con una interfaz gráfica a través de la cual se puede extraer reportes de tráfico de los distintos canales agrupado por hora, día, semana, mes y año, en formato csv y Excel. Estos reportes se utilizan para facturación y por eso es importante que los datos estén actualizados y correctos en todo momento.

La solución actual del CDRManager realiza su procesamiento de forma secuencial. Debido a esto y al creciente tráfico de los servicios de mensajería, el programa se mantiene ocupado procesando CDRs e incluso se presentan retrasos en el procesamiento a causa de no poder cubrir todos los CDRs en el periodo de tiempo actualmente configurado, causando dificultades para los clientes ya que no pueden acceder a la información que desean.

Por lo tanto, se genera la necesidad de implementar un nuevo sistema multihilo aplicando programación paralela para el procesamiento de tráfico de CDRs, con un sistema de generación de reportes que tenga un diseño actual e incluya mejores elementos visuales, utilizando tecnologías orientadas a este propósito.

1.3. Justificación

Hoy en día la mayoría de los sistemas operativos modernos proporcionan características que permiten que un proceso tenga múltiples hilos de control, esto facilita la implementación de programas utilizando técnicas de programación paralela, la cual permite que muchas instrucciones se ejecuten simultáneamente. Utilizar este tipo de programación para el desarrollo de aplicaciones, permite mejorar el desempeño del software de manera general y utilizar de manera eficiente el hardware del computador [13].

Por esta razón, se implementó un nuevo sistema para el procesamiento de tráfico con múltiples hilos con la intención de mejorar el tiempo de procesamiento de CDRs y garantizar la disponibilidad de los datos actualizados.

1.4. Objetivos

1.4.1. Objetivo general

Desarrollar un sistema multihilo para el procesamiento de tráfico y un sistema para la generación de reportes para la compañía Ogangi de Venezuela C.A.

1.4.2. Objetivos específicos

- Realizar investigación sobre programación paralela.
- Diseñar el módulo que se encargará de procesar los CDRs, el módulo que se encargará de generar reportes automáticos, la base de datos y la interfaz WEB para la generación de reportes.
- Implementar el módulo que se encargará de procesar los CDRs, el módulo que se encargará de generar reportes automáticos, la base de datos y la interfaz WEB para la generación de reportes.
- Realizar pruebas de desempeño sobre el nuevo sistema paralelo del CDRManager.
- Realizar pruebas unitarias sobre el sistema web que permite la generación de reportes.

1.5. Metodología

Se utilizó el método ágil SCRUM [4] el cuál es un método ágil que permite llevar el desarrollo del programa de manera iterativa, por medio de ciclos de Sprint (ver la figura 1.1), donde cada ciclo desarrolla un incremento del sistema [7]. Cada sprint dura 2 semanas a lo sumo, e involucra en la actividad de selección de cada Sprint a todas las personas encargadas del proyecto.

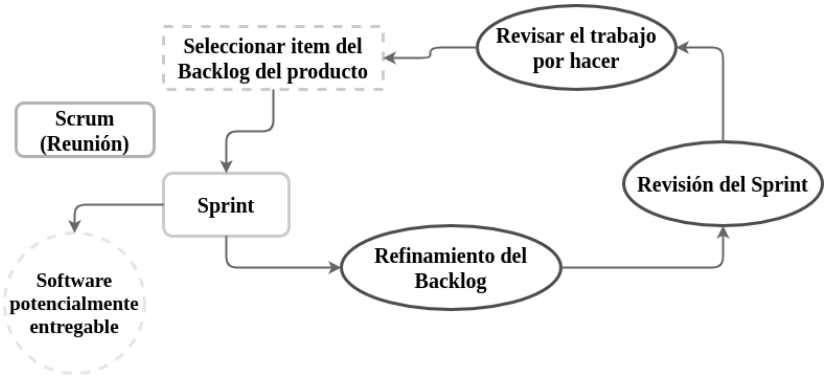

Figura 1.1: Ciclo de Sprint

Está investigación utilizó el siguiente plan de trabajo:

- Analizar y entender el sistema CDRManager actual.
- Revisión de artículos relacionados a la programación paralela.
- Realizar el levantamiento de información y requerimientos del sistema con los principales usuarios del mismo y desde la lógica del sistema actual.
- Investigar sobre las tecnologías que se van a usar para el desarrollo del nuevo sistema.
- Diseñar la Base de Datos.
- Diseñar el módulo por medio de programación paralela para el procesamiento de tráfico.
- Diseñar la interfaz gráfica para el sistema web.
- Implementación del sistema según resultados obtenidos de las actividades de levantamiento de información, análisis y diseño.
- Pruebas generales de funcionalidad sobre el sistema.

1.6. Alcance

En esta investigación se realizó un sistema multihilo por medio de programación paralela para el procesamiento de tráfico y un sistema web para la generación de reportes del proyecto CDRManager. Esta implementación permitió mejorar el tiempo de ejecución para la actualización del tráfico de datos, evitando retrasos en el procesamiento de CDRs; permitiendo a los clientes, por medio del sistema web, realizar los reportes de una manera sencilla.

El sistema multihilo y el sistema web fueron implementados en Spring, el cual es un framework de código abierto y utiliza Java como uno de sus lenguajes de programación. Las vistas para la generación de reportes, se realizaron con Angular 8, de servidor web se utilizó Tomcat y de gestor de Base de Datos se utilizó PostgreSQL.

Capítulo 2

Marco Teórico

La programación paralela es un área de gran interés dentro del campo de la computación, ya que hoy en día las computadoras tienen una arquitectura que permite la realización de tareas de manera paralela. En este capítulo se hablará de los conceptos fundamentales para comprender este trabajo de investigación, extraídos de fuentes reconocidas como lo son: [13] titulado "Fundamentos de Sistemas Operativos", el libro [15] que tiene por nombre "Sistemas Operativos Modernos"; en estos libros se explica de una manera detallada todos los estados de un proceso, de cómo un proceso contiene un hilo de control y de que este a su vez es una unidad básica de utilización de la CPU; estos conceptos son necesarios para entender cómo funcionan los procesadores. Adicionalmente se utilizó [8] titulado "Introducción a la Computación Paralela" para los conceptos de computación paralela, granularidad, grado de paralelismo, procesamiento en paralelo y las clasificaciones de Flyn, donde se explican algunos de los tipos de arquitecturas que existen. Del libro [10] que se titula "Patterns for Parallel Programming", se extrajo la información referente a los patrones de paralelismo que se utilizaron en este proyecto.

También se habla de las tecnologías utilizadas para el desarrollo del proyecto de investigación y de la metodología utilizada para conseguir los objetivos de este proyecto.

2.1. Procesos

Un proceso es un programa en ejecución con un solo hilo de control el cual tiene un contador de programa que especifica la siguiente instrucción que hay que ejecutar y un conjunto de recursos asociados [13, 15]. La diferencia entre un proceso y un programa es muy importante [15] ya que un programa es una entidad pasiva, un archivo que contiene una lista de instrucciones almacenadas en disco (archivo ejecutable), mientras que un proceso es una entidad activa, con un contador de programa que especifica la siguiente instrucción que hay que ejecutar y un conjunto de recursos asociados, estos proporcionan la capacidad de operar con alta concurrencia [13].

Pueden existir dos procesos que estén asociados al mismo programa pero estos se consideran dos secuencias de ejecución separadas ya que las secciones de datos, del cúmulo de memoria y de la pila variarán de unos procesos a otros.

2.1.1. Estado de un proceso

Cuando un proceso es ejecutado, este va cambiando de estado. Estos estados se definen según la actividad actual del proceso.

A continuación se hará una breve explicación de cada uno de dichos estados que han sido extraídos del libro [13]:

- **Nuevo.** El proceso se está creando.
- **En ejecución.** Están siendo ejecutadas las instrucciones.
- **En espera.** El proceso está esperando a que se produzca un suceso (como la terminación de una operación de E/S[2] o la recepción de una señal).
- **Preparado.** El proceso está a la espera de que le asignen a un procesador.
- **Terminado.** Ha terminado la ejecución del proceso.

La mayoría de los procesos terminan cuando concluye su trabajo.

[2]Es un tipo de dispositivo de una computadora capaz de interactuar con los elementos externos a ese sistema de forma bidireccional (permite ingresar información desde un sistema externo, como emitir información a partir de ese sistema).

2.2. Hilos

Un hilo es una unidad básica de utilización de la unidad central de procesamiento (CPU, por sus siglas en inglés) de una computadora; tiene un identificador, un contador de programa, un conjunto de registros y una pila [13]. En la actualidad, la mayoría de los sistemas operativos permiten que un proceso tenga múltiples hilos de control puesto que es mas eficiente utilizar un proceso que contenga múltiples hilos que crear un proceso nuevo que va a realizar las mismas tareas de procesos existentes ya que esto lleva tiempo y hace uso intensivo de los recursos; estos pueden compartir el espacio de direcciones y todos sus datos entre ellos [13, 15].

El uso de la programación con múltiples hilos permite que un programa continúe ejecutándose aunque parte de él esté bloqueado o realizando una operación muy larga; los hilos comparten la memoria y los recursos del proceso al que pertenecen. En las arquitecturas con multiprocesadores, las ventajas de usar multihilos se incrementan porque es posible que los hilos se ejecuten en paralelo en los diferentes procesadores.

2.3. Computación paralela

La computación paralela permite descomponer un problema en subproblemas; estos pueden ejecutarse al mismo tiempo de manera segura, permitiendo estructurar el código para que los subproblemas se ejecuten simultáneamente y de esta manera aprovechar la concurrencia [10]. Gracias a los avances en campos tan diversos como el arquitectónico, las tecnologías de interconexión, los ambientes de programación, entre otros, se han desarrollado computadoras paralelas. También hay que reconocer qué áreas de aplicación podrían beneficiarse con el uso de computadoras paralelas, ya que pueden llegar a producir resultados más rápidos [8].

Existen muchos esquemas de clasificación, el más popular es la taxonomía de Flyn ya que es la clasificación utilizada en computación paralela, este utiliza ideas familiares al campo de la computación convencional para proponer una taxonomía de arquitecturas de computadoras.

La información a continuación se extrajo del libro “Introducción a la computación paralela“ [8], siguiendo la secuencia del mismo para nombrar las clasificaciones.

2.3.1. Clasificación de Flyn

Esta clasificación está basada en el número de flujos de instrucciones (n_i) y de datos (n_d) simultáneos que pueden ser tratados por el sistema computacional durante la ejecución de un programa; siendo el flujo de instrucción una secuencia de instrucciones transmitidas desde una unidad de control a uno o más procesadores, y el flujo de datos una secuencia de datos que viene desde un área de memoria a un procesador y viceversa.

2.3.1.1. Secuencia de instrucción única, secuencia de datos única

Esta categoría se conoce como SISD (por sus siglas en inglés) y tiene un único programa que es ejecutado usando solamente un conjunto de datos específicos a él; está compuesto de una memoria central donde se guardan los datos y los programas, y de un procesador (unidad de control y unidad de procesamiento). Este tipo de maquina representa la clásica máquina de Von-Neumann donde $n_i = n_d = 1$. En esta plataforma sólo se puede dar un tipo de paralelismo virtual a través del paradigma de multitareas, donde el tiempo del procesador es compartido entre diferentes programas. Más que paralelismo, lo que soporta esta plataforma es un tipo de concurrencia.

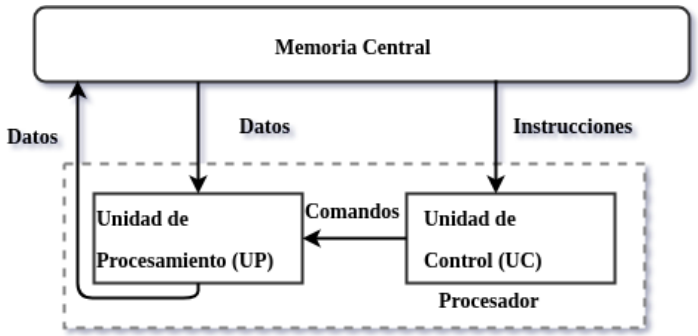

Figura 2.1: Arquitectura tipo SISD [8].

2.3.1.2. Secuencia de instrucción única, Secuencia de datos múltiple

Conocida como SIMD (por sus siglas en inglés), es un arreglo de elementos de procesamiento que ejecutan la misma instrucción al mismo tiempo. En este caso $n_i = 1$ y $n_d > 1$. En estas arquitecturas, un controlador recibe y decodifica secuencias de instrucciones a ejecutar, para después enviarlas a múltiples procesadores esclavos. El

arreglo de procesadores procesa los datos que llegan a los diferentes procesadores, usando la instrucción enviada por el controlador. Los procesadores están conectados a través de una red. Los datos a tratar pueden estar en un espacio de memoria que es común a todos los procesadores o en un espacio de memoria propio a cada unidad de procesamiento (ver figura 2.2).

Un controlador envía instrucciones desde la memoria a un conjunto de procesadores, así cada procesador es simplemente una unidad aritmética-lógica y se tiene una sola unidad de control. Todos los procesadores ejecutan cada operación recibida al mismo tiempo; por lo tanto, cada uno de los procesadores ejecuta la misma instrucción sobre diferentes datos. Los sistemas SIMD tienen una CPU adicional para procesamiento escalar, de tal manera de mezclar operaciones entre el arreglo de procesadores y el procesador secuencial estándar; esto es debido a que en la mayoría de las aplicaciones paralelas se tienen fases de código secuencial y de código paralelo. De esta forma, la parte de código secuencial es ejecutado en la CPU adicional. Esto también implica dos procesos diferentes para mover los datos entre el arreglo de procesadores y el procesador secuencial: en un caso se difunden los datos desde el procesador secuencial al arreglo de procesadores; en el otro caso, al moverse los datos del arreglo de procesadores al procesador secuencial, los datos son reducidos.

El programador no notará que las instrucciones del programa se ejecutan de manera múltiple sobre diferentes datos y no una sola vez, por lo que para él sigue siendo una maquina secuencial.

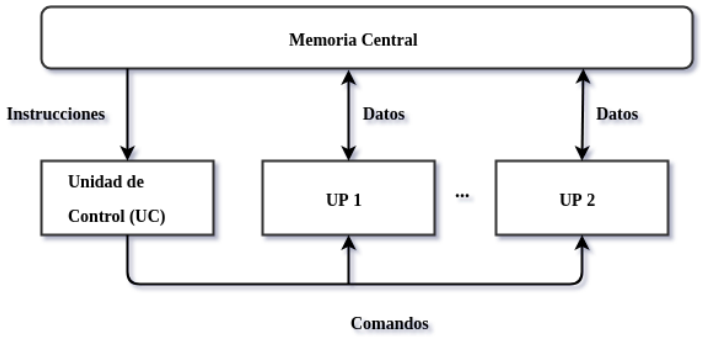

Figura 2.2: Arquitectura tipo SIMD [8].

2.3.2. Flujo de instrucción múltiple, flujo de datos único

Este sistema se conoce como MISD (por sus siglas en inglés). Estas son computadoras con elementos de procesamiento, cada uno ejecutando una tarea diferente, de tal forma que todos los datos a procesar deben ser pasados a través de cada elemento de procesamiento para su procesamiento (ver Figura 2.3). En este caso $n_i > 1$ y $n_d = 1$. Este tipo de computadoras no corresponde a un modo de funcionamiento realista.

La idea es descomponer las unidades de procesamiento en fases, en donde cada una se encarga de una parte de las operaciones a realizar. De esta manera, parte de los datos pueden ser procesados en la fase uno mientras otros son procesados en la fase dos, otros en la tres, y así sucesivamente. El flujo de información es continuo y la velocidad de procesamiento crece con las etapas.

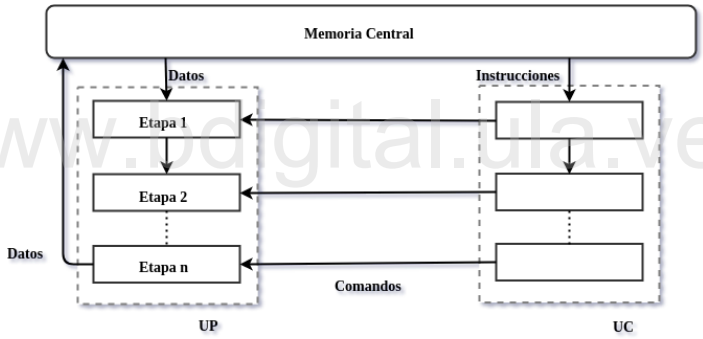

Figura 2.3: Arquitectura tipo MISD [8].

2.3.2.1. Flujo de instrucción múltiple, flujo de datos múltiple

El sistema MIMD (por sus siglas en inglés) es el modelo mas general de paralelismo, y debido a su flexibilidad, una gran variedad de tipos de paralelismo pueden ser explotados. Las ideas básicas son que múltiples tareas heterogéneas puedan ser ejecutadas al mismo tiempo, y que cada procesador opere independientemente con ocasionales sincronizaciones con otros. Está compuesto por un conjunto de elementos de procesamiento donde cada uno realiza una tarea, independiente o no, con respecto a los otros procesadores. La forma de programación usualmente utilizada es del tipo concurrente, en la cual múltiples tareas que pueden ser diferentes entre ellas, se pueden

ejecutar simultáneamente. En este caso $n_i > 1$ y $n_d > 1$. En esta categoría existen muchos sistemas de multiprocesadores y sistemas de múltiples tareas. Una computadora MIMD es fuertemente acoplada si existe mucha interacción entre los procesadores, de lo contrario es débilmente acoplada.

Hay dos tipos de distinciones para las computadoras MIMD, están los sistemas con múltiples espacios de direcciones (MIMD a memoria distribuida) y los sistemas con espacios de direcciones compartidos (MIMD a memoria compartida).

- **Memoria Compartida:**

  En este tipo de arquitectura el conjunto de microprocesadores comparten la misma memoria. Cada procesador puede ejecutar una instrucción diferente y el acceso a la memoria se hace a través de una red de interconexión (ver figura 2.4). La memoria se puede dividir en varios módulos, y en un momento dado, a un módulo de memoria solo lo puede acceder un procesador. En este tipo de computadoras, ejecutar un código secuencial es muy sencillo. La complejidad de la red de interconexión aumenta rápidamente al aumentar el número de procesadores, ya que todos deben poder acceder de manera eficaz todos los bancos de memoria, lo que limita su uso a un número reducido de procesadores. Al poseer pocos procesadores, se utilizan procesadores que sean poderosos, de tal manera que la potencia de la máquina paralela esté dada por la potencia de los procesadores más que por el número de ellos.

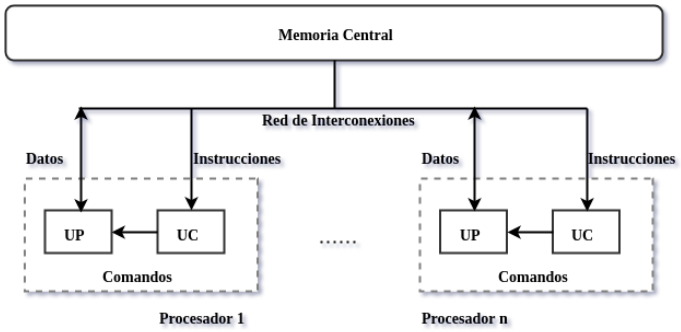

Figura 2.4: Arquitectura tipo MIMD con Memoria Compartida [8].

La comunicación entre los procesadores se hace a través de la memoria, por lo que

se necesita una programación rigurosa a nivel de la organización de la memoria para evitar conflictos de acceso a una misma localidad de memoria. El objetivo de rendimiento es mover los datos a usar a la memoria antes de que sean requeridos y tiene que ver con el hecho de que entre más cerca estén los datos del procesador, más rápido serán accedidos. El problema está en a quién sacar y cuándo mover los datos, de tal manera que cuando un procesador los necesite, ya los datos estén en un sitio de acceso rápido.

- **Memoria Distribuida:**

  En estas computadoras cada procesador es autónomo, con las mismas características de una máquina secuencial. En este caso se replica la memoria, los procesadores y los controladores, así cada procesador tiene su propia memoria local (ver figura 2.5). Los procesadores se pueden comunicar entre ellos a través de una red de interconexión, la cual puede tener diferentes formas. Los procesadores se comunican por pase de mensajes. Los rendimientos de estas máquinas están muy ligados a los rendimientos de las comunicaciones de las máquinas, y el paralelismo a utilizar debe ser explícito, lo que hace requerir nuevos sistemas de explotación y ambientes de programación.

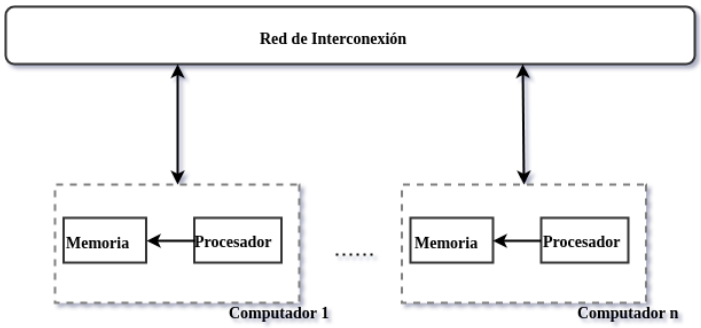

Figura 2.5: Arquitectura tipo MIMD con Memoria Distribuida [8]

2.3.3. Procesamiento en paralelo

El procesamiento en paralelo es una forma de procesamiento que permite la ejecución de varios procesos concurrentemente [8] agilizando el rendimiento de la CPU

[15]. Este tipo de procesamiento surge de la necesidad que hay de producir resultados más rápidos los cuales tardarían mucho con un procesamiento secuencial.

2.3.4. Granularidad

El grano del paralelismo es la cantidad procesamiento que necesita realizar una tarea antes de comunicarse con otra [8].

Existen dos tipos de granularidad, grano fino y grano grueso. Una granularidad fina, tiene más comunicación y cambios de contextos entre las tareas, siendo una aplicación con comunicación intensiva; una granularidad de grano grueso o fuerte, tiene menos comunicación, pero las tareas que son concurrentes pueden quedar agrupadas y ser ejecutadas secuencialmente. Así, la determinación del grano ideal, es importante para definir la mejor relación entre paralelismo (grano fino) y comunicación (grano grueso o fuerte).

2.3.5. Grado de paralelismo

El grado de paralelismo se define como el número de tareas que se pueden ejecutar en paralelo durante la ejecución de una aplicación. Este puede variar a través de las diferentes secciones de un programa, por lo que se habla de un grado de paralelismo máximo, mínimo y promedio, siendo el máximo el límite superior en cuanto al número de procesadores que podrían ser utilizados [8].

2.4. Patrones de diseño

Un patrón de diseño es una técnica que describe una solución a un problema que es común en un contexto particular el cual puede volver a ser utilizado; normalmente se utilizan en el área de diseño de software. Estos patrones siguen ciertas características; un nombre para el patrón, un descripción del patrón, los objetivos y las restricciones [10].

Los patrones de diseño no solo se utilizan en el área de diseño de software sino que también se han extendido hacia otras áreas, una de ellas es el área de programación

paralela, esta área ya tiene desarrollado varios patrones de diseños para el desarrollo de aplicaciones paralelas

2.4.1. Patrones de paralelismo

Los patrones de paralelismo son diseñados para que el programador pueda usarlos como guía y así realizar de una manera mas fácil un programa en paralelo [17].

En esta investigación se utilizaron algunos de los patrones de paralelismo que están contenidos en el libro [10], a continuación se hará una breve explicación extraída del mismo de cada uno de los patrones utilizados para resolver el problema de esta investigación siguiendo la estructura del libro:

2.4.1.1. Patrón de la descomposición de tareas

1. **Problema:** ¿Cómo se puede descomponer un problema en tareas que pueden ejecutarse simultáneamente?.

2. **Contexto:** El programador debe comprender cuáles son las partes del problema que son intensivas en computación, luego debe definir las tareas que conforman el problema y elaborar un algoritmo que permita ejecutar simultáneamente estas tareas. Puede que el problema se divida en una colección de tareas que serán casi independientes. En algunos casos es difícil separar la división de tareas de la división de datos por lo que es difícil definir el problema y por esta razón el programador debe volver a analizar el problema.

3. **Compensaciones:**

   - Flexibilidad. Permite adaptarse a diferentes implementaciones.
   - Eficiencia. Permite alcanzar mejores rendimientos.
   - Simplicidad. Permite poder mantener el código con un esfuerzo razonable.

4. **Solución:** Las tareas deben descomponerse de manera tal que sean suficientemente independientes para que la gestión de las dependencias tome

solo una pequeña fracción del tiempo total de ejecución del programa (ver la figura 2.6).

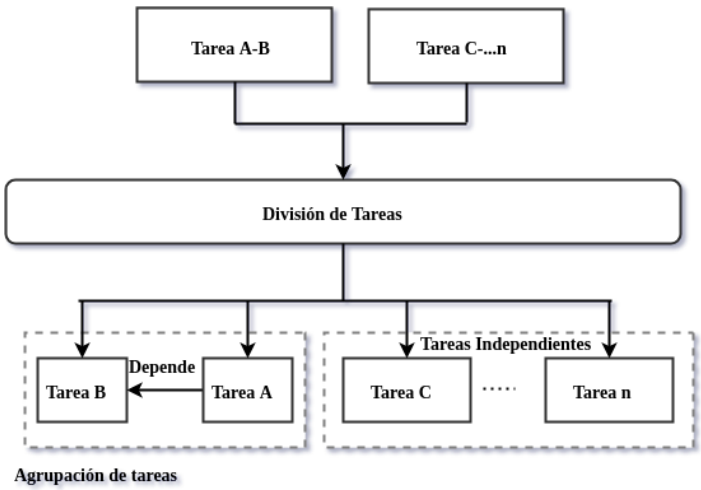

Figura 2.6: Descomposición de Tareas

2.4.1.2. Patrón de agrupación de tareas

Luego de haberse dividido las tareas, es necesario poder reagruparlas.

1. **Problema:** ¿Cómo se pueden reagrupar las tareas para simplificar el manejo de las dependencias?.

2. **Contexto:** Este patrón se puede aplicar luego de haber dividido las tareas. El patrón describe el primer paso para analizar las dependencias entre las tareas dentro de la descomposición de un problema. Las agrupaciones de tareas simplifican el análisis de dependencias de un problema. Si un grupo comparte una restricción temporal, dicha restricción se puede aplicar de una vez para todo el grupo. Si un grupo de tareas debe trabajar en conjunto en una estructura de datos compartida, la sincronización requerida se puede establecer una vez para todo el grupo. Si un conjunto de tareas es independiente, combinarlas en un solo grupo y programarlas para que se ejecuten como un solo grupo grande puede simplificar el diseño y aumentar la concurrencia disponible. En cada caso, la idea

es definir grupos de tareas que compartan restricciones y simplifiquen el problema de administrar restricciones al tratar con grupos en lugar de tareas individuales.

3. **Solución:** Hay varios tipos de dependencias entre tareas, estas pueden agruparse de la siguiente manera:

   - Dependencia temporal. Es una restricción en el orden en que se ejecuta un conjunto de tareas. Si la tarea A depende del resultado de la tarea B, esta sólo puede ejecutarse cuando la tarea B haya finalizado.
   - Tareas que se ejecutan simultáneamente. Estas deben ejecutarse al mismo tiempo para evitar bloqueos o interrupciones ya que algunas tareas esperan datos de otras tareas inactivas.
   - En algunos casos estas tareas son completamente independientes entre si; esto hace que puedan ejecutarse en cualquier orden, incluso al mismo tiempo. Es importante tener en cuenta cuando esto se cumple.

El objetivo de este patrón es poder agrupar tareas dependiendo de las restricciones, proporcionando una flexibilidad adicional para programar la ejecución de las tareas y una mejor escalabilidad (ver la figura 2.7).

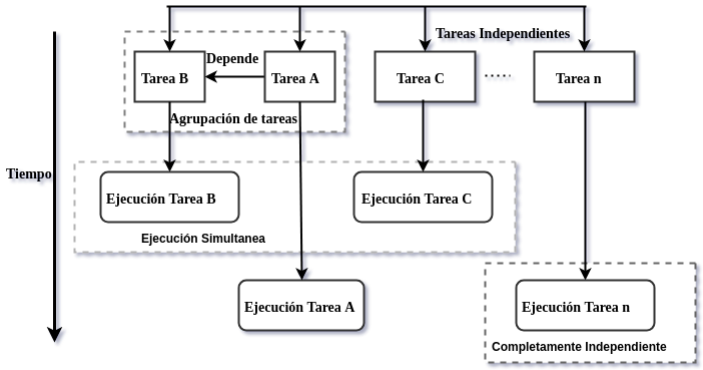

Figura 2.7: Tipos de Agrupación de Tareas

2.4.1.3. Patrón de descomposición de datos

1. **Problema:** ¿Cómo se pueden dividir los datos de un problema en unidades para puedan ser procesados de una manera relativamente independiente?.

2. **Contexto:** Luego de que el programador tiene una comprensión clara del problema que está resolviendo, debe identificar las partes computacionales más intensas del problema para luego considerar la descomposición de los datos que están en las tareas que conforman el problema. También se debe definir si descomponer por datos o por tareas ya que tanto las tareas como la descomposición de los datos deben tomarse en cuenta para crear un algoritmo paralelo. Una descomposición basada inicialmente en datos es mejor si la parte donde está el cómputo intensivo del problema está organizado en torno a la manipulación de un gran volumen de datos y si se están aplicando operaciones similares a diferentes partes de la estructura de datos, de tal manera que las diferentes partes pueden operarse de forma relativamente independiente.

3. **Compensaciones:**

   - Flexibilidad. Permite adaptar el diseño a diferentes requisitos de implementación.
   - Eficiencia. Permite alcanzar mejores rendimientos.
   - Simplicidad. Permite poder mantener el código con un esfuerzo razonable.

4. **Solución:** Si ya se ha realizado la división por tareas, la descomposición de los datos se debe a las necesidades de cada tarea. La descomposición debe ser simple si se pueden asociar datos bien definidos y distintos a cada tarea. Si se comienza con una descomposición de datos, se debe analizar las estructuras de datos centrales que definen el problema y hay que considerar si se pueden dividir en partes para operar concurrentemente.

   Al considerar descomponer las estructuras de datos se tiene que tener en cuenta el tamaño y la cantidad de fragmentos de datos ya que estos deben ser flexibles. Una manera de conseguirlo es definiendo fragmentos cuyo tamaño y número están controlados por un pequeño número de parámetros. Es importante que los fragmentos sean lo suficientemente grandes para que la cantidad de trabajo al actualizar los fragmentos compensen la sobrecarga de la gestión de dependencias.

2.4.1.4. Patrón de distribución de datos

1. **Problema:** Dada una descomposición de datos para un problema. ¿Cómo se comparten los datos entre las tareas?.

2. **Contexto:** Luego de haber agrupado las tareas, el siguiente paso es analizar cómo se comparten los datos entre grupos de tareas para que el acceso a los datos compartidos se pueda administrar correctamente. El enfoque en esta etapa del análisis se desplaza a la descomposición de los datos, cada una asociada con una o más tareas que manejan la actualización de una porción de datos que a veces se denomina datos locales, porque está relacionada con las tareas responsables de su actualización. Normalmente las tareas no funcionan con sólo sus propios datos locales; los datos pueden necesitar ser compartidos entre las tareas.

3. **Compensaciones:** La distribución de datos puede tener importantes implicaciones tanto para la correctitud del cómputo como para la eficiencia.

   - Si el intercambio se realiza incorrectamente, una tarea puede obtener datos no válidos debido a una condición de carrera [3], donde una tarea puede leer desde una ubicación de memoria antes de que se complete la escritura de los datos esperados.
   - Garantizar que los datos compartidos estén listos para su uso, puede generar una sobrecarga excesiva de sincronización.
   - Hay que tener en cuenta el costo comunicacional al compartir los datos. Con frecuencia, una mejor opción es estructurar el algoritmo y las tareas de modo que se minimice la cantidad de datos compartidos para comunicarse.

     Entonces, el objetivo es permitir compartir datos sin interferir con la eficiencia.

4. **Solución:** El primer paso es identificar los datos que se comparten entre las

[3] Es un error que ocurre cuando la salida o estado de un proceso es dependiente de una secuencia de eventos que se ejecutan en orden arbitrario y van a trabajar sobre un mismo recurso compartido. Al no ejecutarse los eventos en el orden correcto pueden provocar inconsistencias y comportamientos impredecibles y no compatibles con un sistema determinista

tareas. Es necesario analizar los datos para ver en qué categoría se encuentran y de esta manera ver como se utilizan. Estas categorías son:

- Sólo lectura. Los datos se leen pero no se escriben. El acceso a estos valores no necesita estar protegido.
- Lectura-Escritura. Se accede a los datos mediante más de una tarea. Cualquier acceso a los datos (Escritura o lectura) debe estar protegido con algún tipo de mecanismo con acceso exclusivo(bloqueos, semáforos, etc.), que puede ser muy costoso. Algunas clasificaciones que se le dan a estos casos son:
- Acumulaciones: Los datos se utilizan para acumular un resultado. Para cada ubicación en los datos compartidos, los valores se actualizan mediante múltiples tareas, y la actualización se realiza a través de algún tipo de operación de acumulación asociativa, puede ser por medio de suma, mínima y máxima pero se puede usar cualquier operación asociativa en pares de operandos. Para los datos, cada tarea tiene una copia local separada que luego se acumulan en una sola copia global al final de la acumulación.
- Lectura múltiple/Escritura individual: Los datos se leen mediante múltiples tareas, cuyas tareas necesitan su valor inicial pero se modifican con una sola tarea, que puede leer o escribir su valor de forma arbitraria. Para datos de este tipo, se necesitan al menos dos copias, una para conservar el valor inicial y otra para la tarea de modificación; la copia que contiene el valor inicial se puede descartar cuando ya no sea necesaria.

2.4.1.5. Patrón Maestro/Trabajador

1. **Problema:** ¿Cómo debe organizarse un programa cuando el diseño está dominado por la necesidad de equilibrar las cargas de trabajo dinámicamente?

2. **Contexto:** La carga debe estar balanceada para lograr una mayor eficiencia. En ocasiones es difícil equilibrar la carga de trabajo. Algunas de las causas que dificultan el equilibrio de la carga de trabajo están a continuación:

- Las cargas de trabajo son variables e impredecibles. Esto hace que no se puedan clasificar en contenedores de igual coste causando que la distribución no sea optima.
- Cuando las tareas comparten datos de lectura y escritura y deben estar activas al mismo tiempo. En este caso el patrón no es aplicable.

Este patrón es aplicable si las tareas son independientes entre sí, o si las dependencias son débiles; el programador tiene una flexibilidad mucho mayor en cómo equilibrar la carga. Esto permite que el balanceo de carga se realice automáticamente.

3. **Compensaciones:**

   - Las predicciones explícitas del tiempo de ejecución para cualquier tarea dada no son posibles y el diseño debe equilibrar la carga sin ellas.
   - Las operaciones para equilibrar las cargas imponen una sobrecarga de comunicación y pueden ser muy costosas. Esto sugiere que la programación debe girar en torno a un número menor de tareas grandes. Sin embargo, estas reducen la cantidad de formas en que se pueden dividir las tareas, lo que dificulta el logro de un buen equilibrio de carga.
   - La lógica para crear un algoritmo de balanceo puede ser complicado y puede requerir cambios por errores en el programa.

4. **Solución:** La solución consta de un maestro y una o mas instancias de un trabajador. El maestro es el encargado de iniciar el cálculo y configurar el problema, luego crea una bolsa de tareas. El trabajador toma una tarea de la bolsa de tareas, realiza el trabajo indicado, prueba su finalización y luego va a buscar la siguiente tarea. Esto continúa hasta que se cumple la condición de terminación, momento en el cual el maestro se despierta, recopila los resultados y finaliza el cálculo (ver la figura 2.8).

   Los algoritmos Maestro/Trabajador equilibran automáticamente la carga. Esta decisión es tomada dinámicamente por el maestro cuando un trabajador completa una tarea y accede a la bolsa de tareas para trabajar más.

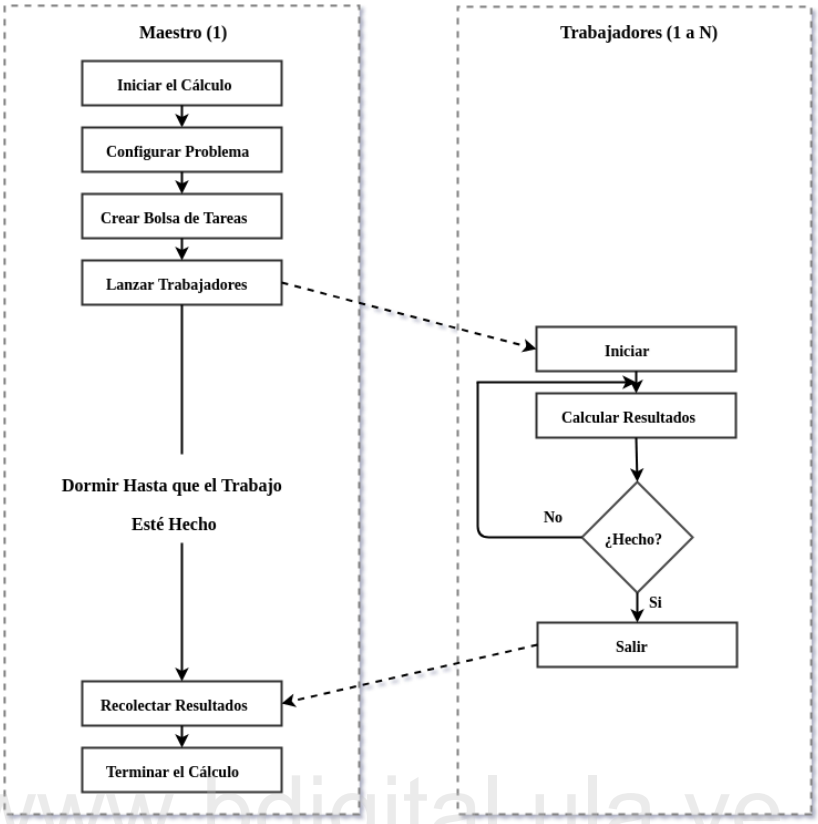

Figura 2.8: Patrón Maestro/Trabajador [10].

2.5. Angular

Angular es una plataforma y un framework para la creación de aplicaciones cliente en HTML y TypeScript, desarrollado en TypeScript, de código abierto, mantenido por Google; utilizado para crear y mantener aplicaciones web [2].

Angular se basa en clases tipo “Componentes”. Este es un bloque, que contiene una plantilla HTML con estilos y funcionalidad.

2.6. JAVA

Java es un lenguaje de programación orientado a objetos, portable y multiplataforma. Este permite ejecutar un mismo código en cualquier Sistema Operativo gracias a que utiliza un entorno de ejecución de JAVA llamado JRE [6].

Este lenguaje fue desarrollado por Sun Microsystems, posteriormente adquirido por Oracle. En la actualidad puede utilizarse de modo gratuito, pudiéndose conseguir sin problemas un paquete para desarrolladores que oriente la actividad de programar en este lenguaje.

Como ya se mencionó anteriormente, JAVA ha sido utilizado en muchos desarrollos de Programación de Alto Desempeño (HPC) y gracias a las clases sobre manejo de hilos que trae, se puede utilizar para el desarrollo de programas en paralelo de una manera fácil.

2.7. Spring Framework

Spring Framework es utilizado para programar y configurar aplicaciones empresariales modernas basadas en JAVA, en cualquier tipo de plataforma de implementación [5].

2.8. PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto, que utiliza y amplía el lenguaje SQL combinado con muchas características que almacenan y escalan de forma segura las cargas de trabajo de datos más complicadas [1]. Este gestor es gratuito y de código abierto.

Muchas de las funciones requeridas por el estándar SQL son compatibles, aunque a veces con una función o una sintaxis ligeramente diferente. A partir de la versión 11, PostgreSQL cumple con al menos 160 de las 179 funciones obligatorias para SQL:2011 Core conformance. A partir de este escrito, ninguna base de datos relacional cumple con la conformidad total con este estándar.

Capítulo 3

Análisis de Requisitos

Realizar la recolección de los requisitos es necesario para poder cumplir con las necesidades del usuario, logrando un sistema de calidad y por ende satisfacción en los usuarios. Los primeros pasos que deben seguirse para recolectar los requisitos correctamente es dividir en dos listas las peticiones de los usuarios, estas listas se convierten en requisitos funcionales y no funcionales. Luego, con la lista de los requisitos funcionales, se realiza la definición de los actores que se utilizaran en el sistema y las funcionalidades que tendrá cada actor en el mismo, siguiendo con los diagramas de actividades y de secuencia los cuales explican de una manera gráfica y sencilla las funcionalidades mas importantes.

3.1. Ingeniería de requisitos

La Ingeniería de Requisitos es una rama de la Ingeniería del Software que se encarga de la compresión de las necesidades exactas de los usuarios de un sistema y de esta manera convierte estas necesidades en requisitos que serán usadas en el desarrollo del mismo. Estos requisitos son fundamentales en el desarrollo del sistema ya que se enfoca en lo que se desea producir, porque describe con claridad el comportamiento del sistema y de esta manera se logra obtener un sistema de calidad que satisface las necesidades del usuario.

Esta lista de requisitos se divide en dos. Los requisitos funcionales y no funcionales.

3.1.1. Requisitos funcionales

Los requisitos funcionales describen los comportamientos o funciones que el sistema deba realizar.

1. Gestión de reportes.
2. Gestión de Usuarios.
3. Gestión de la Base de datos.
4. Gestión de permisos de administrador a usuarios.

3.1.2. Requisitos no funcionales

Los requisitos no funcionales son las cualidades que el sistema debe tener imponiendo restricción en el diseño del sistema.

1. La interfaz debe ser intuitiva para el usuario.
2. Tener acceso a las diferentes funciones del sistema dependiendo del rol del usuario.
3. El software se realizará con el sistema operativo Linux, de servidor web Tomcat, PostgreSQL como sistema de gestor de Base de Datos. Se utilizará el framework Spring y como lenguaje de programación, Java.
4. El sistema web debe tener un manual de usuario para la creación de reportes.

3.2. Especificación de requisitos

3.2.1. Actores del sistema CDRManager

1. **Usuario no Autorizado:** Son todas las personas que pueden ingresar al sistema para conocer la información que se muestra públicamente. Estos actores pueden pasar a ser usuarios autorizados si el administrador los registra en la aplicación web con los roles de gestor, empleado o cliente.

2. **Usuario Autorizado:**

   - **Gestor:** El gestor puede ser un empleado o un cliente con permisología dada por el administrador para gestionar la aplicación web.
   - **Cliente:** El cliente es el encargado de generar los reportes con el tráfico de datos respectivo a su compañía.
   - **Empleado:** El empleado puede generar reportes de cualquiera de los clientes que utilizan el servicio de mensajería corporativa que ofrece la compañía.

3. **Administrador:** El administrador del sistema, es aquella persona que maneja y tiene acceso a todos los datos en el sistema.

   - Se encarga de registrar a una persona como cliente o empleado en el sistema.
   - Es capaz de asignar permisos extras al cliente y empleado para que estos puedan gestionar algunas funcionalidades del sistema.
   - Puede eliminar a los usuarios autorizados del sistema.
   - Gestiona la información de la Base de Datos.

3.2.2. Jerarquía de actores

Primero se debe recordar que un usuario no autorizado, es un usuario que no ha iniciado sesión, este puede o no poseer una cuenta registrada, un usuario autorizado es aquel que posee una cuenta y ha iniciado sesión, y por último un administrador es un usuario que se encarga de administrar el sistema.

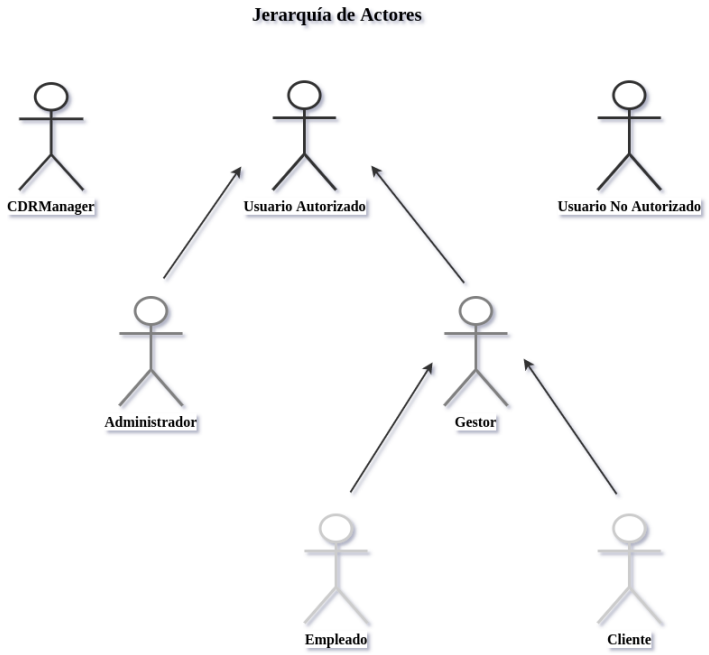

Figura 3.1: Jerarquía de Actores.

El rol CDRManager es el encargado de actualizar cada cierto tiempo, la base de datos por medio de procesamiento en paralelo con el fin de mantener la información actualizada para que los usuarios autorizados puedan gestionar los reportes sin tener retrasos con la información que necesiten del tráfico de datos.

3.2.3. Casos de uso

En la figura(3.2), se pueden apreciar los respectivos casos de uso en un mismo diagrama.

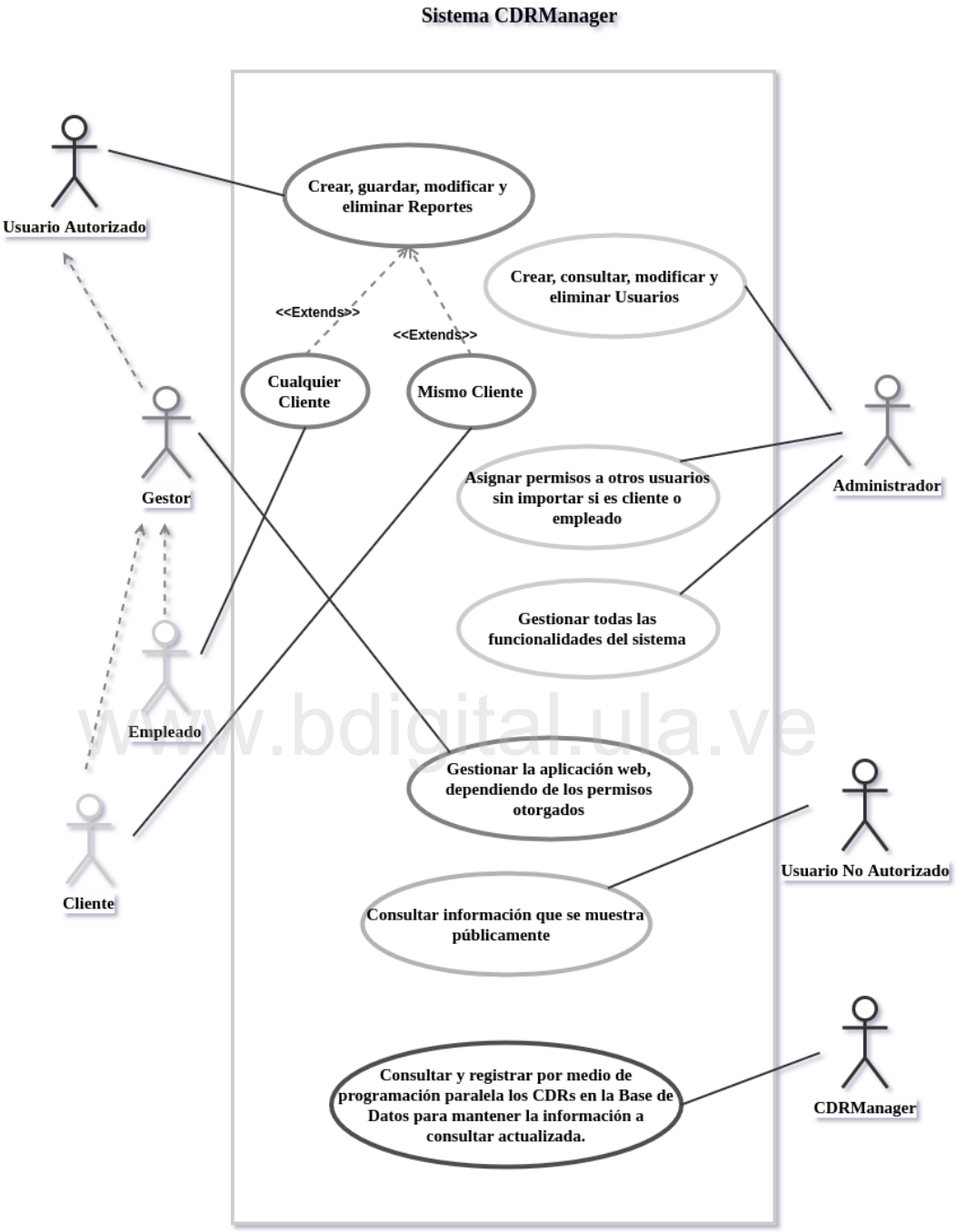

Figura 3.2: Casos de Uso.

A continuación se presentan de manera independiente los casos de uso para un mejor detalle.

3.2.3.1. Usuario CDRManager

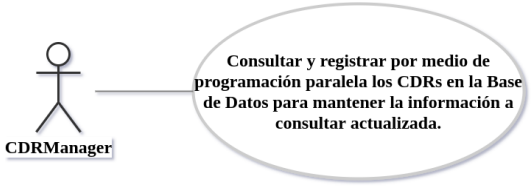

Figura 3.3: Caso de Uso del rol CDRManager.

3.2.3.2. Usuario no autorizado

Figura 3.4: Caso de Uso del usuario No Autorizado.

3.2.3.3. Usuario autorizado

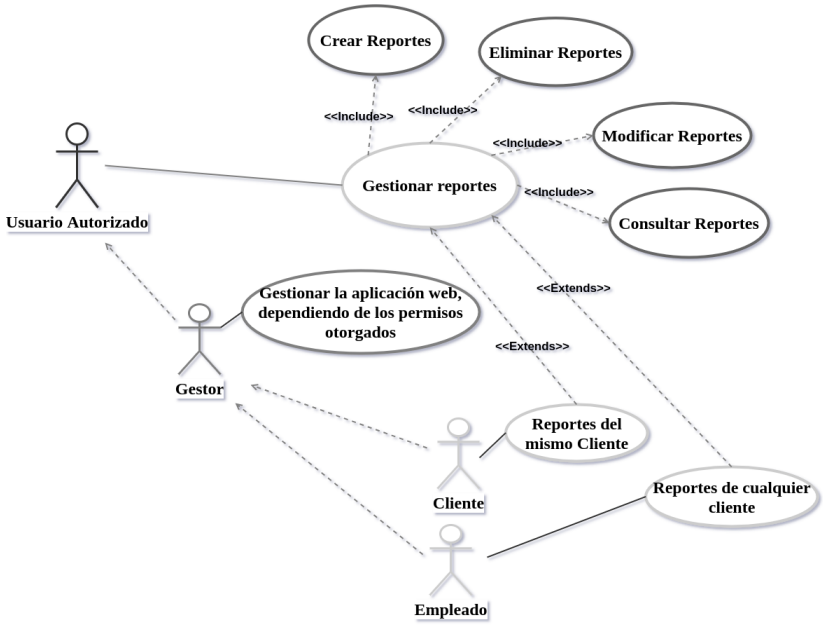

Figura 3.5: Caso de Uso del usuario Autorizado.

3.2.3.4. Administrador

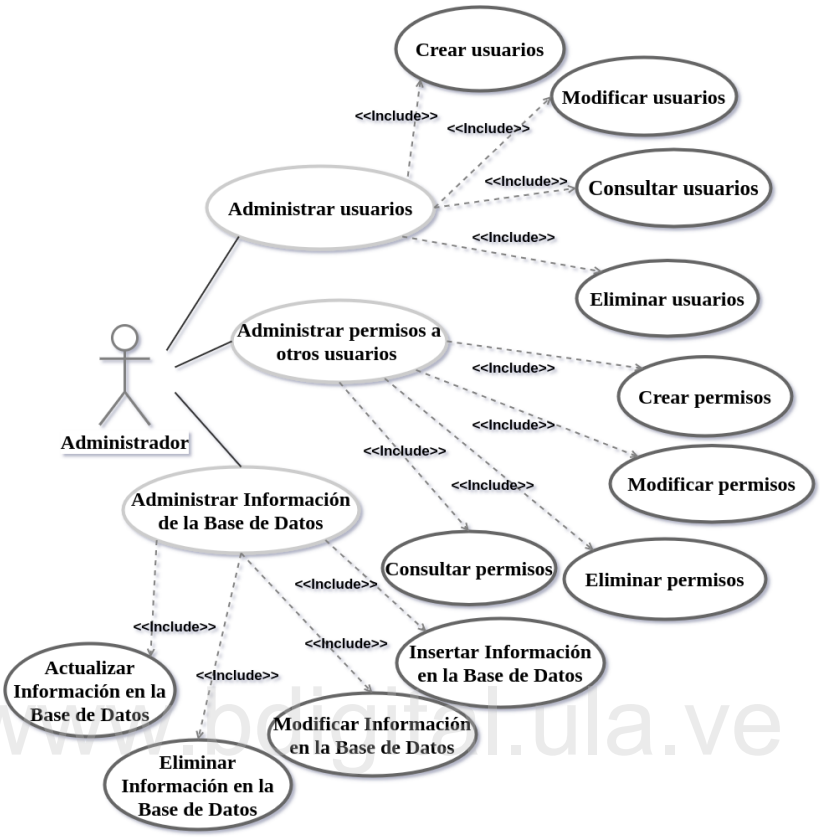

Figura 3.6: Caso de Uso del Administrador.

3.2.3.5. Gestor

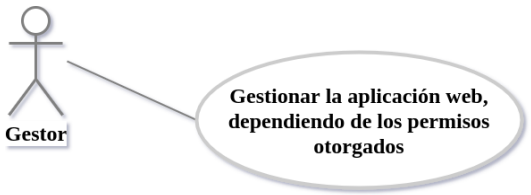

Figura 3.7: Caso de Uso del Gestor.

3.2.4. Diagramas de actividades

Los siguientes diagramas representan básicamente la realización de operaciones y transiciones entre actividades, representando el flujo de control necesario entre ellas,

para completar la operación.

3.2.4.1. Procesamiento del tráfico de CDRs

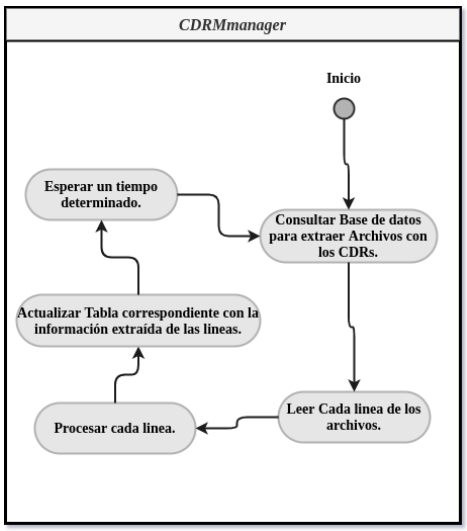

Figura 3.8: Diagrama de Actividades. Procesamiento de tráfico de CDRs.

3.2.4.2. Iniciar sesión

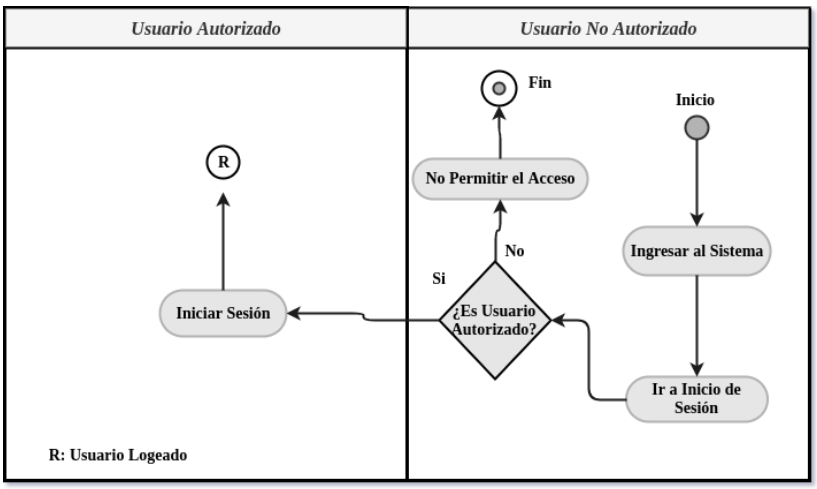

Figura 3.9: Diagrama de Actividades. Iniciar Sesión.

3.2.4.3. Registrar usuario

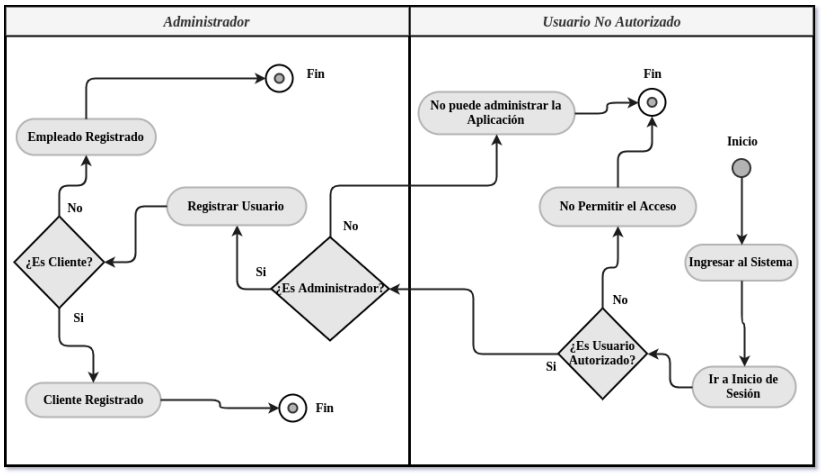

Figura 3.10: Diagrama de Actividades. Registrar Usuario.

3.2.4.4. Asignar permisos para gestionar la aplicación

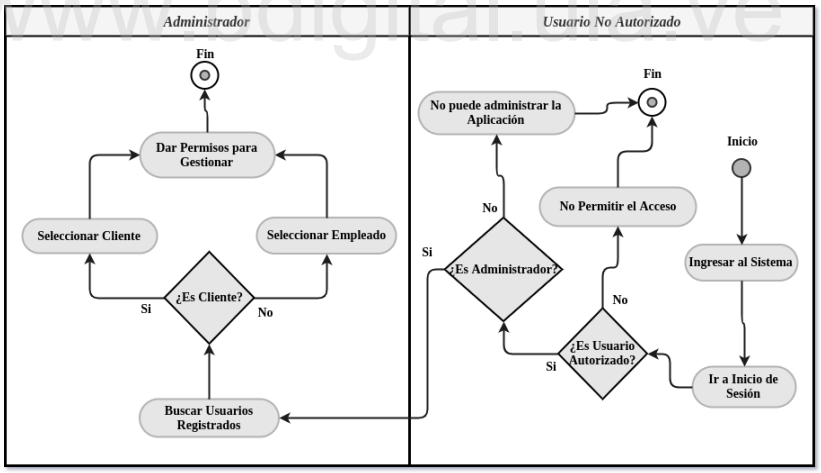

Figura 3.11: Diagrama de Actividades. Asignar Permisos.

3.2.4.5. Crear reporte

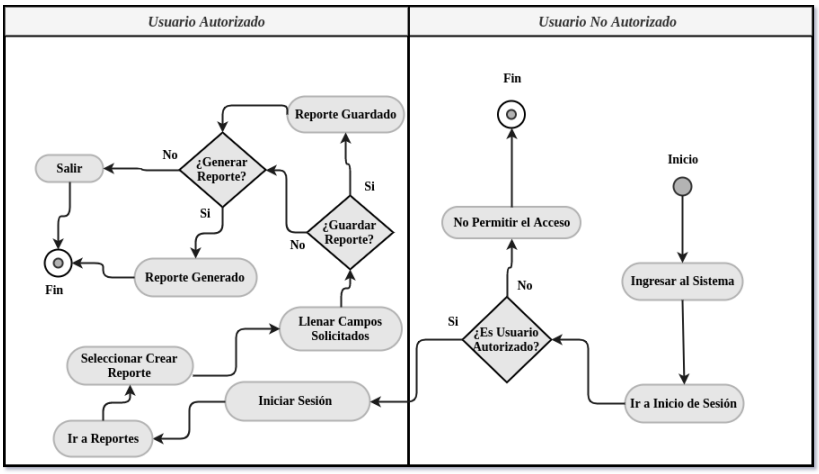

Figura 3.12: Diagrama de Actividades. Crear Reporte.

3.2.5. Diagrama de secuencia

El diagrama de secuencia es un tipo de diagrama de interacción que describe el comportamiento dinámico del sistema mostrando los mensajes intercambiados por los objetos.

Se crearon los diagramas de las funcionalidades mas importantes del sistema que son: Iniciar sesión, registrar usuario, el procesamiento de tráfico del CdrManager y crear el reporte.

3.2.5.1. Iniciar sesión

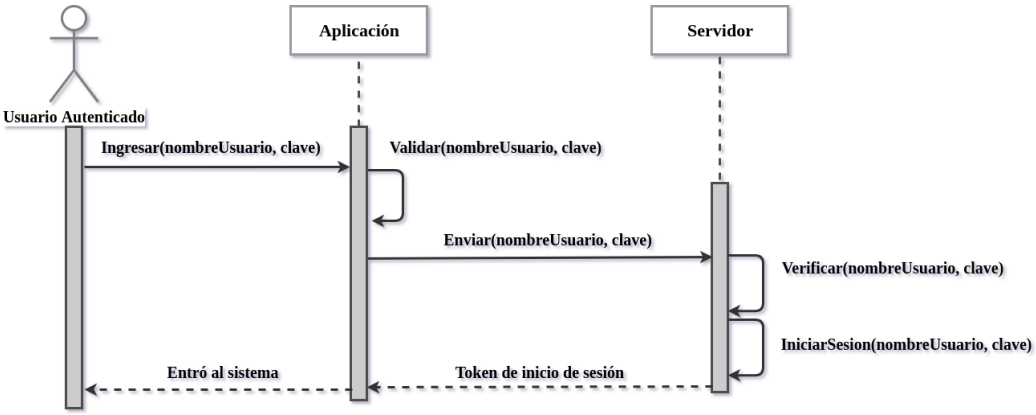

Figura 3.13: Diagrama de secuencia. Iniciar sesión.

3.2.5.2. Registrar usuario

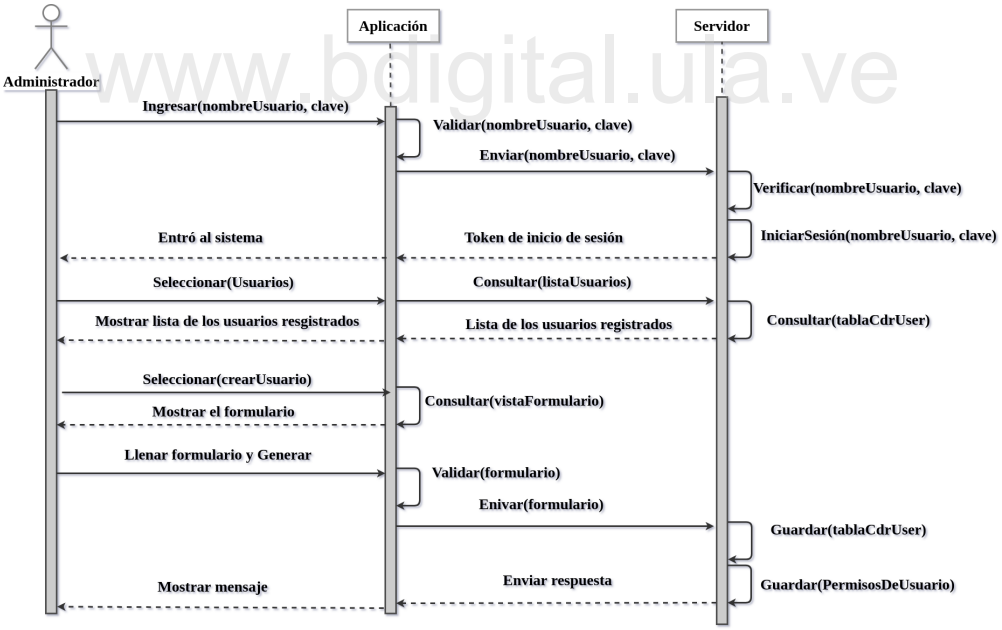

Figura 3.14: Diagrama de secuencia. Registrar usuario.

3.2.5.3. Procesamiento de tráfico del CDRManager

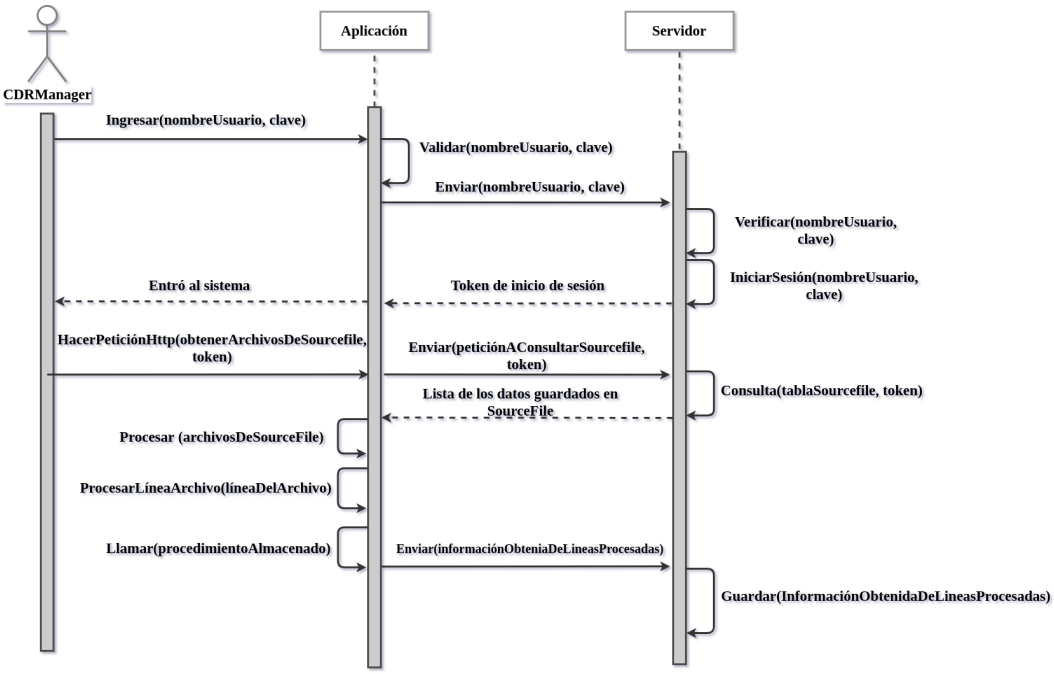

Figura 3.15: Diagrama de secuencia. Procesamiento de tráfico.

3.2.5.4. Crear reporte

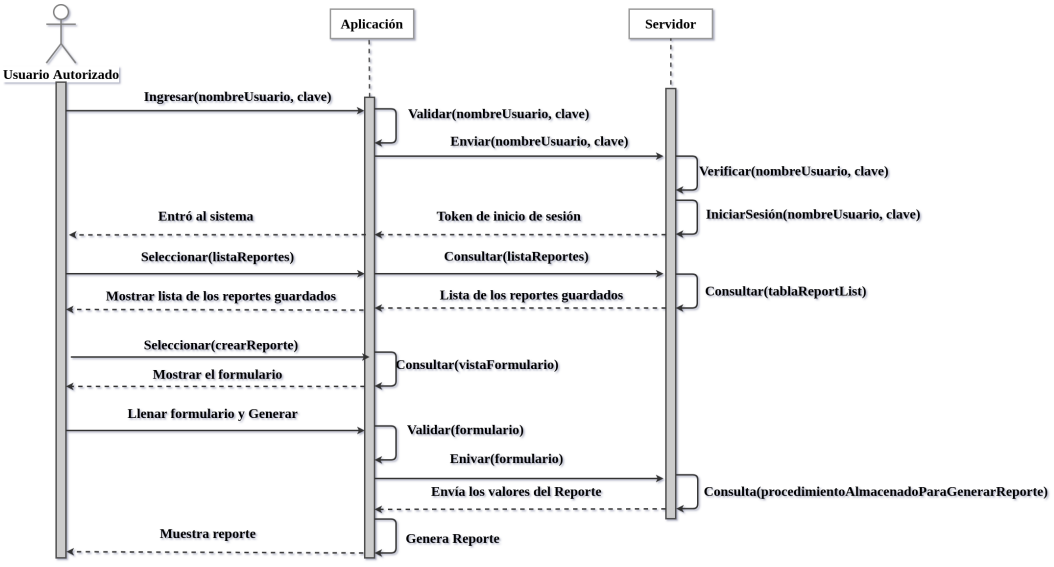

Figura 3.16: Diagrama de secuencia. Crear Reporte.

Capítulo 4

Diseño

El diseño permite plasmar la apariencia y funcionalidades que tendrá cada uno de los módulos de este proyecto de grado, por esta razón en este capítulo se mostrará el diseño de la arquitectura del sistema que se va a utilizar, el diseño que se implementará de la base de datos que será utilizado por el módulo del procesamiento en paralelo tanto en un diagrama secuencial como de clases, seguido de los patrones de diseños utilizados para lograr un diseño eficiente del módulo de procesamiento en paralelo. Luego se podrá apreciar un diagrama donde se observa la manera en como se puede generar un reporte y por último el diseño de las vistas de la aplicación que permitirá la generación de los reportes automáticos.

4.1. Arquitectura del sistema

La arquitectura del sistema se divide en dos niveles, cliente-servidor y al mismo tiempo es una arquitectura de tres capas, ya que se utiliza el paradigma MVC (Modelo, Vista, Controlador), bajo este modelo, la dinámica completa del sistema, se gestiona a través del controlador, comunicándose este, tanto con la vista como con el modelo. Ver la figura (4.1)

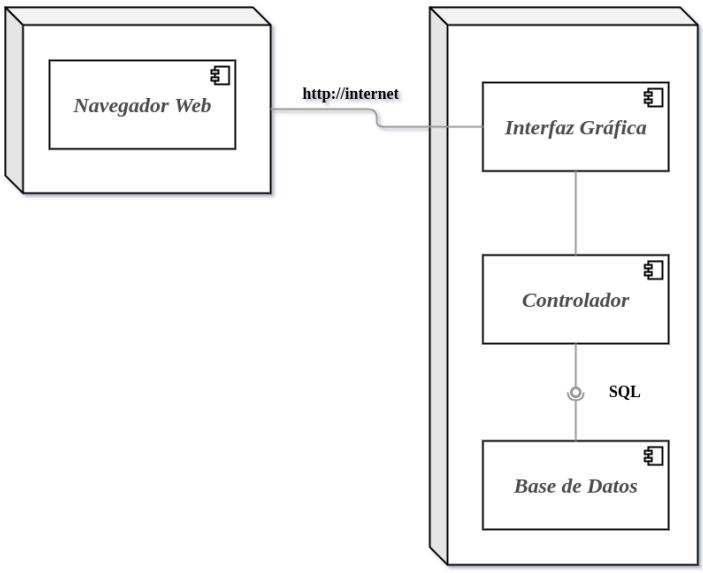

Figura 4.1: Diagrama de despliegue de la arquitectura del sistema.

4.2. Diseño de la base de datos

Para realizar el diseño de la base de datos del CDRManager, se analizó la base de datos del sistema anterior, la cual es una base de datos relacional que cuenta con 144 tablas y tiene una estructura Data Warehouse. Luego de analizar dicha base de datos, esta se redujo a 57 tablas, ya que muchas tablas no se utilizaban y muchos campos de estas 57 tablas no eran necesarias. El diseño nuevo de la base de datos del proyecto no tuvo mucha variación, sigue siendo una base de datos Data Warehouse. Para el procesamiento en paralelo, se utilizan 2 tablas, $mtmo\_hourly$ y $sourcefile$.

En la tabla $sourcefile$ se encuentran guardadas las direcciones de los archivos que contienen el tráfico de datos y la información necesaria para identificar que tipo de CDR es el que contiene el archivo. En $mtmo\_hourly$, se guardan las líneas que han sido procesadas de los respectivos archivos nombrados anteriormente, estos archivos son de tipo de mensaje MT y MO, donde MT son los mensajes enviados por la compañía Ogangi, por orden de los clientes, a sus usuarios y los MO son los mensajes enviados por los usuarios, al cliente y son procesados por la empresa.

En la figura (4.2), se puede apreciar el diseño conceptual de la base de datos que

se utiliza en el módulo que se encargará de procesar los CDRs y en la figura (4.3) el diagrama de clases.

4.2.1. Diagrama ERE

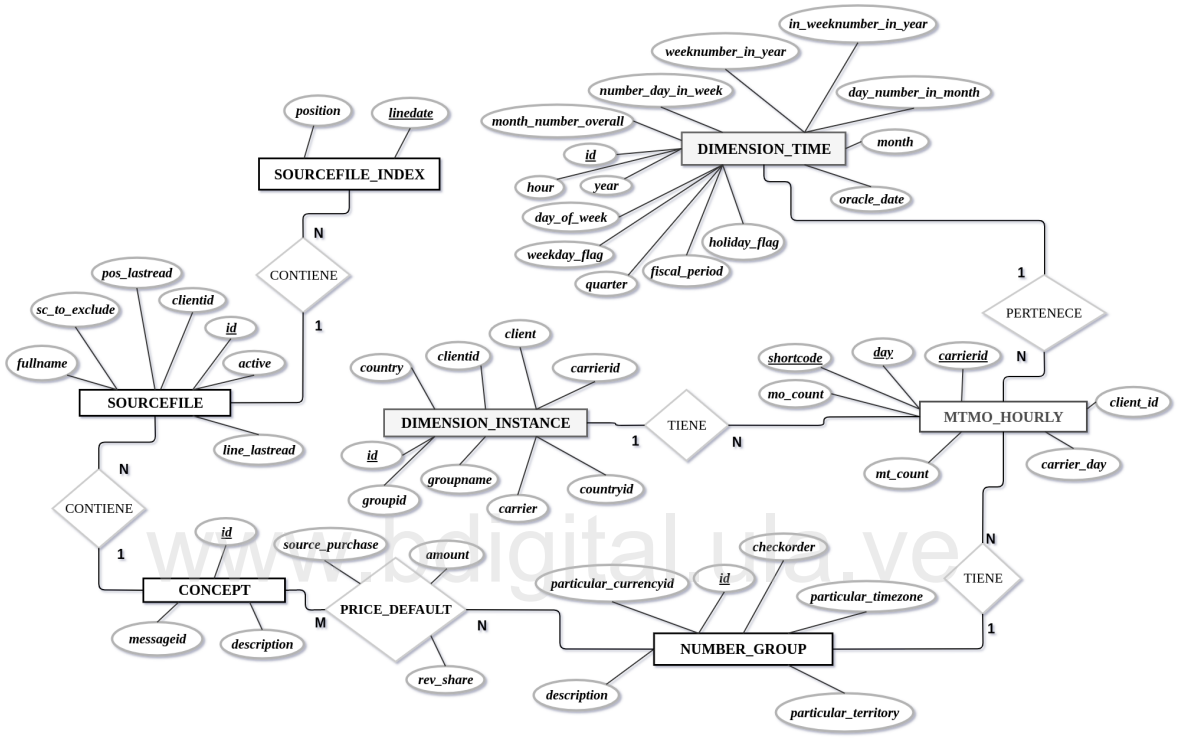

Figura 4.2: Diagrama Conceptual de la Base de Datos utilizada en el Procesamiento en Paralelo

4.2.2. Diagrama de clases

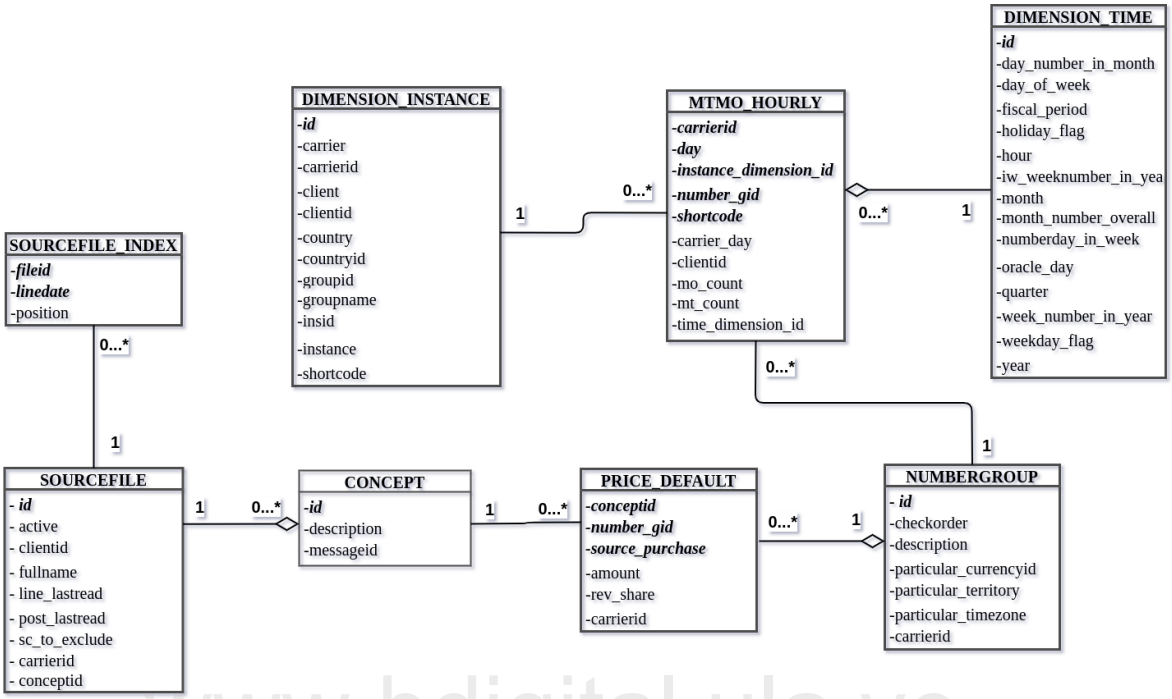

Figura 4.3: Diagrama de clases de la Base de Datos utilizada en el Procesamiento en Paralelo

4.3. Diseño del módulo para el procesamiento de CDRs

4.3.1. Patrones de Diseños utilizados

Para el diseño del módulo que se encarga del procesamiento de CDRs se utilizaron el patrón de descomposición de tareas (ver figura 2.6), el patrón de agrupación de tareas (ver figura 2.7) y el patrón Maestro/Trabajador (ver figura 2.8). El patrón de descomposición de tareas, muestra como dividir las tareas de una forma donde no haya dependencia entre ellas y de esta manera resulta más fácil dividir las tareas entre los hilos para ser procesadas, el patrón de agrupación de tareas analiza diferentes tipos de dependencias que existen entre las tareas y como pueden agruparse y el patrón

Maestro/Trabajador permite balancear los archivos que van a ser procesados por los hilos. En la figura 4.4 se aprecia el diseño del balanceo de los archivos consultados de la tabla sourcefile para luego ser distribuidos a cada hilo, estos archivos son completamente independientes. Los hilos al tener su lista de archivos, procesan cada archivo e insertan la línea procesada en la Base de datos, específicamente en la tabla mtmo_hourly (ver figura 4.2)

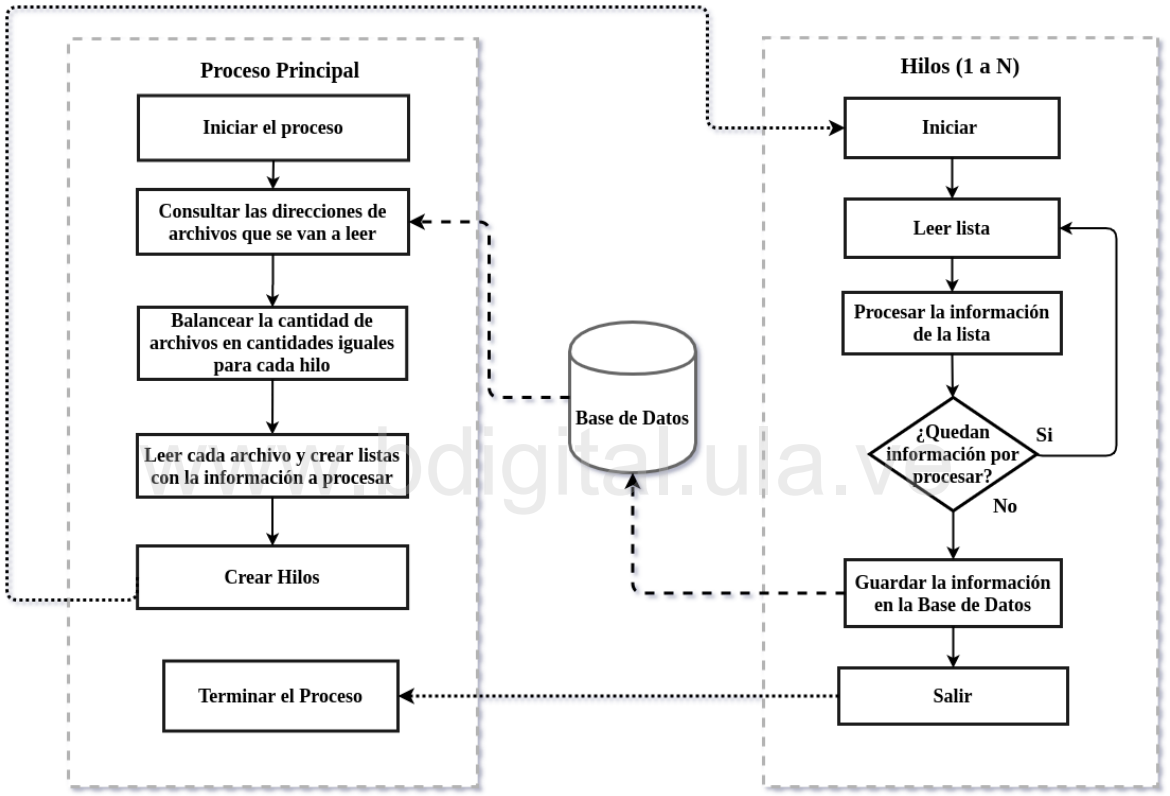

Figura 4.4: Balanceo de Archivos

4.4. Diseño del módulo para generar reportes automáticos

El diseño del módulo para generar reportes automáticos se realiza de manera sencilla para que el usuario autorizado pueda comprender. Este diseño se basa en crear en la interfaz un formulario que le permita al usuario autorizado seleccionar la información

que desea en el archivo para enviar a la base de datos y generar el archivo.

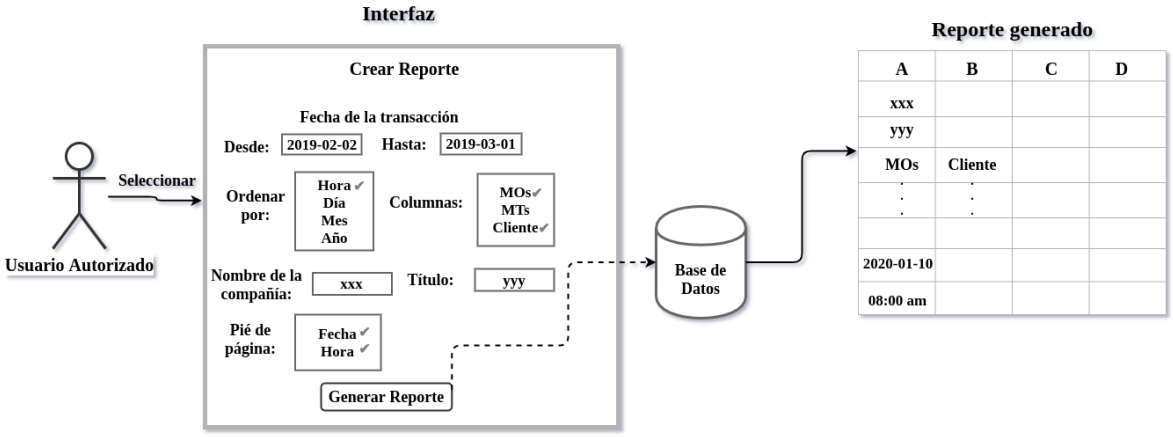

Figura 4.5: Reportes Automáticos

La información solicitada para la generación de reportes, es la fecha desde y hasta donde se quiere consultar la información para el archivo, la agrupación de la información la cual puede ser por Hora, Día, Semana, Mes y Año; la información que desea consultar (mensajes de texto Mo, los Mt y el cliente a quien pertenece la información), un nombre de la compañía, un título para el reporte y la información que desea en el pié de página, esta puede ser la fecha y/o la hora actual. Al ya haber ingresado toda la información, el reporte se envía a la base de datos para realizar la consulta y luego generar el reporte con la información solicitada.

4.5. Diseño de la interfaz WEB

Se diseñó la interfaz de una manera sencilla para una mejor comprensión del usuario. El diseño de cada vista se muestra a continuación:

- La figura 4.6 permite que el usuario autenticado puede ingresar al sistema CDRManager.

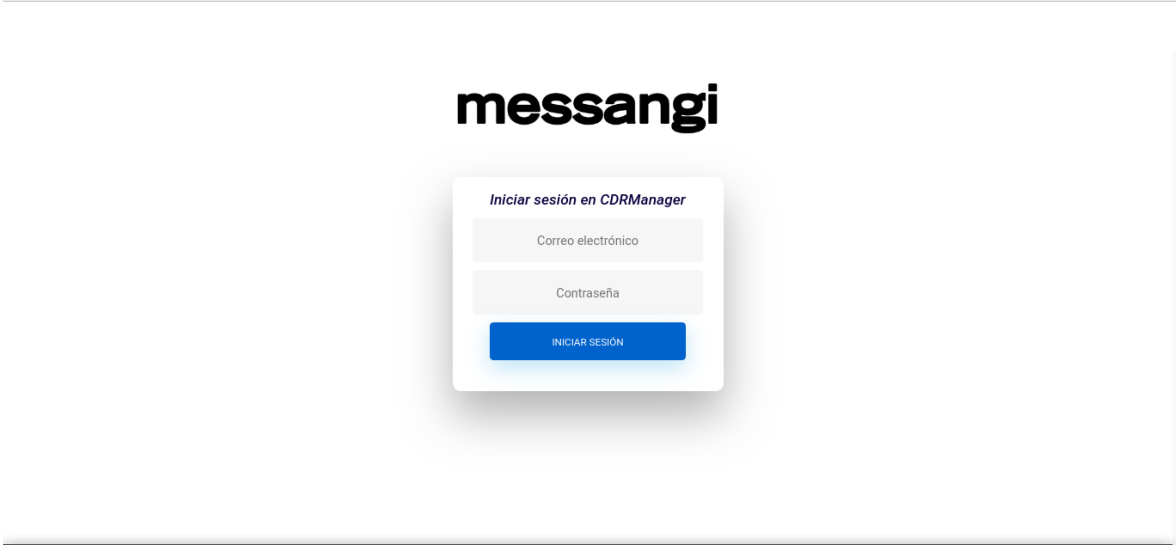

Figura 4.6: Inicio de sesión del CDRManager.

- La figura 4.7 muestra la página principal para los usuarios con rol cliente o empleado.

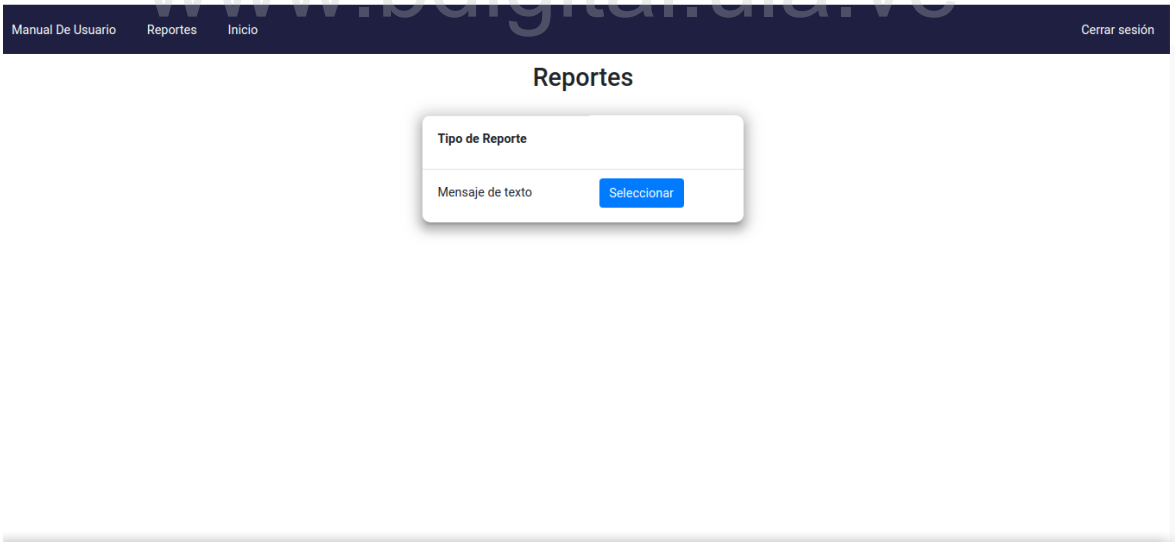

Figura 4.7: Página de inicio para cliente y empleado.

- La figura 4.8 muestra la página principal para el administrador y para el rol gestor.

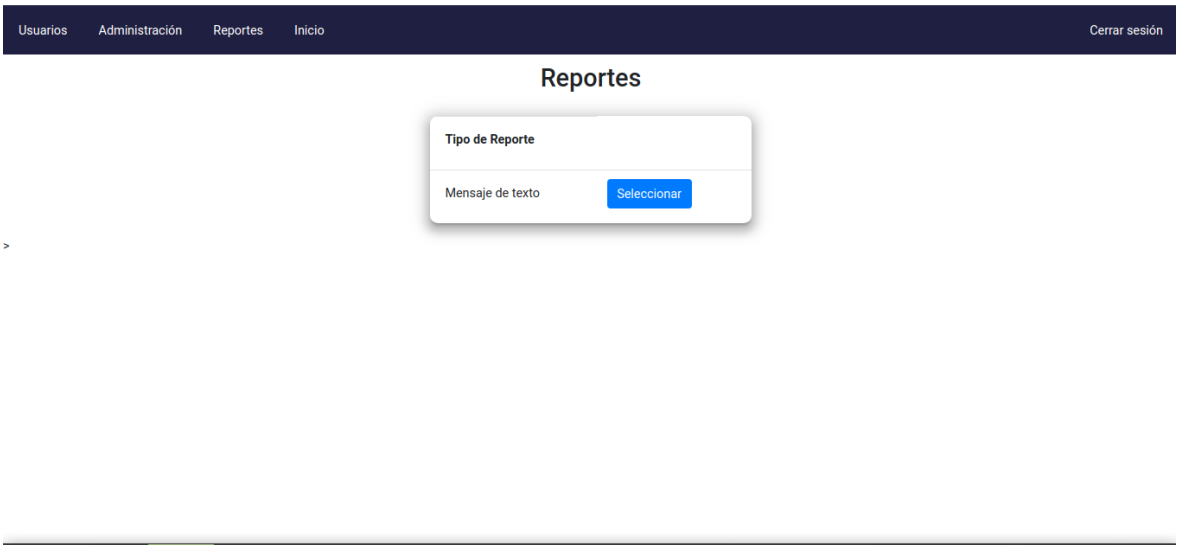

Figura 4.8: Página de inicio para administrador y gestor.

- En la figura 4.9 se muestra la página donde se pueden ver los usuarios registrados en el sistema.

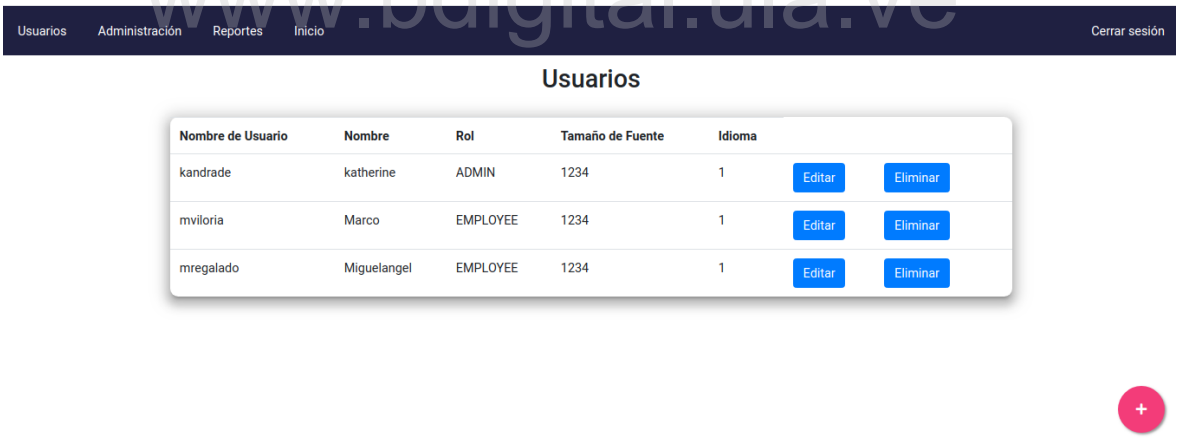

Figura 4.9: Vista para gestionar los usuarios registrados en el sistema.

- En la figura 4.10 se encuentra la vista que permite agregar usuarios al sistema para que puedan hacer uso del mismo. Los administradores y los usuarios que tengan el rol gestor y el permiso otorgado por el administrador para gestionar los usuarios, son los únicos que pueden agregar personas en el sistema.

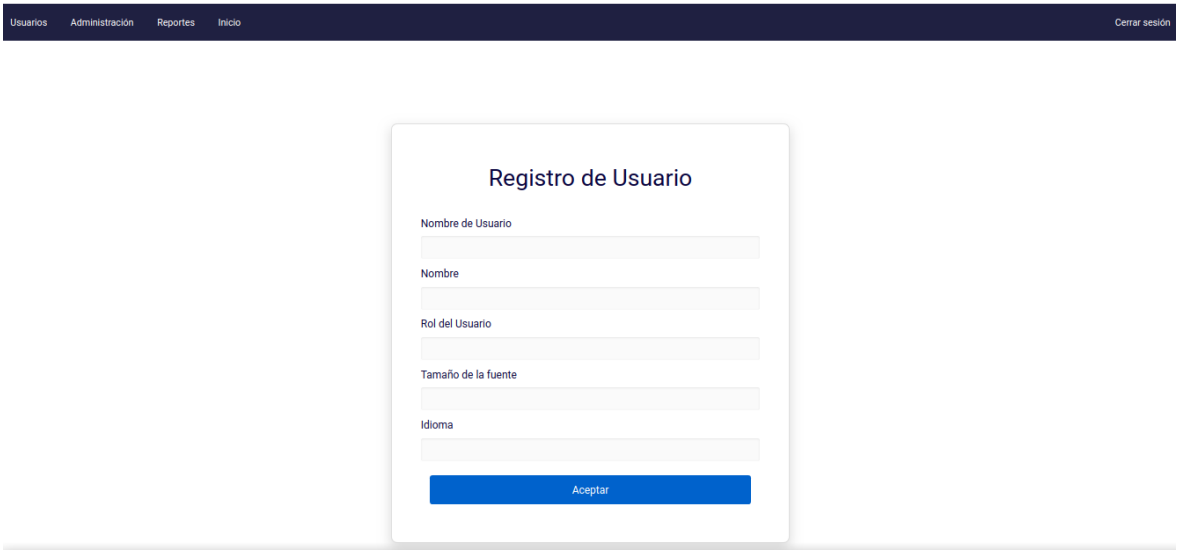

Figura 4.10: Vista para la creación de los usuarios que utilizarán el sistema.

- En la figura 4.11 se observa la vista para editar los usuarios del sistema.

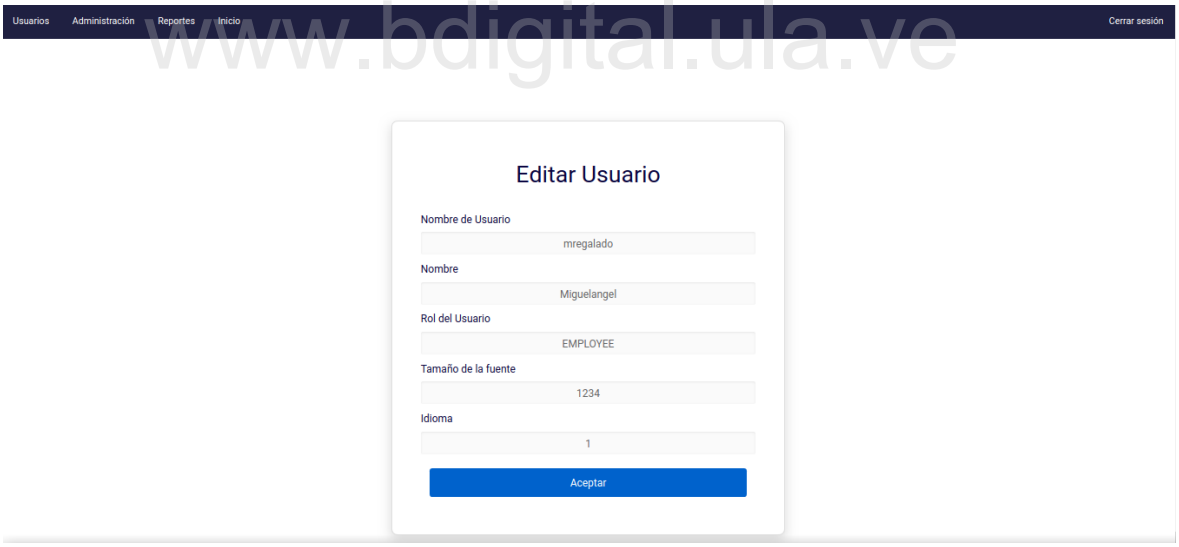

Figura 4.11: Vista para editar los usuarios que se encuentran registrados en el sistema.

- En la figura 4.12 se muestra la página de los listados de los tipos de mensaje que maneja la compañía.

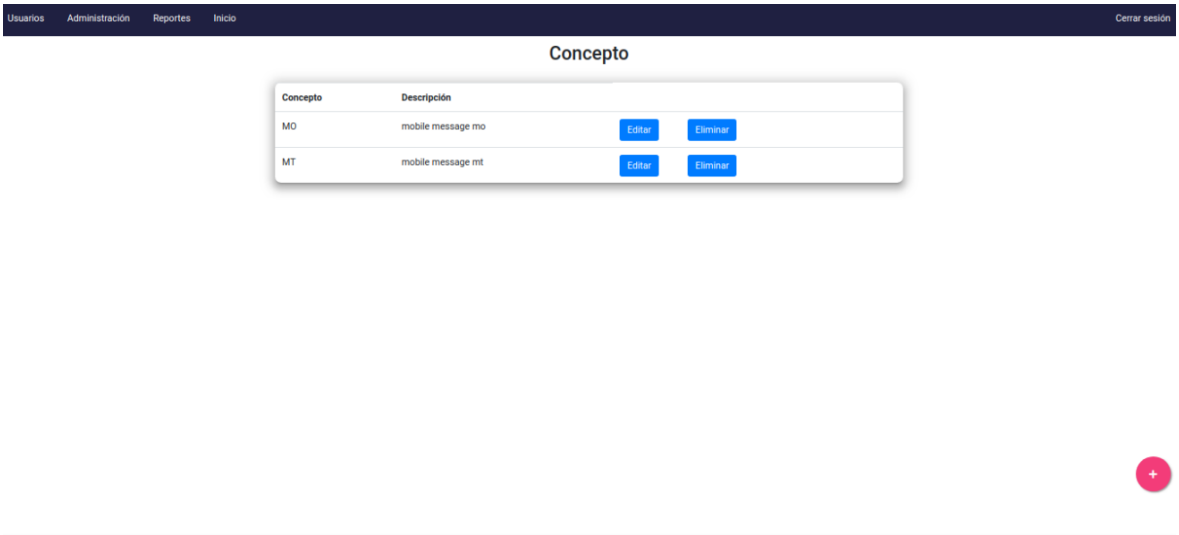

Figura 4.12: Vista de la lista de conceptos que existen en el sistema.

- La figura 4.13 muestra como se ingresan los conceptos al sistema.

Figura 4.13: Vista para la creación de los conceptos que manejará el sistema.

- En la figura 4.14 se observa la vista para editar los conceptos del sistema.

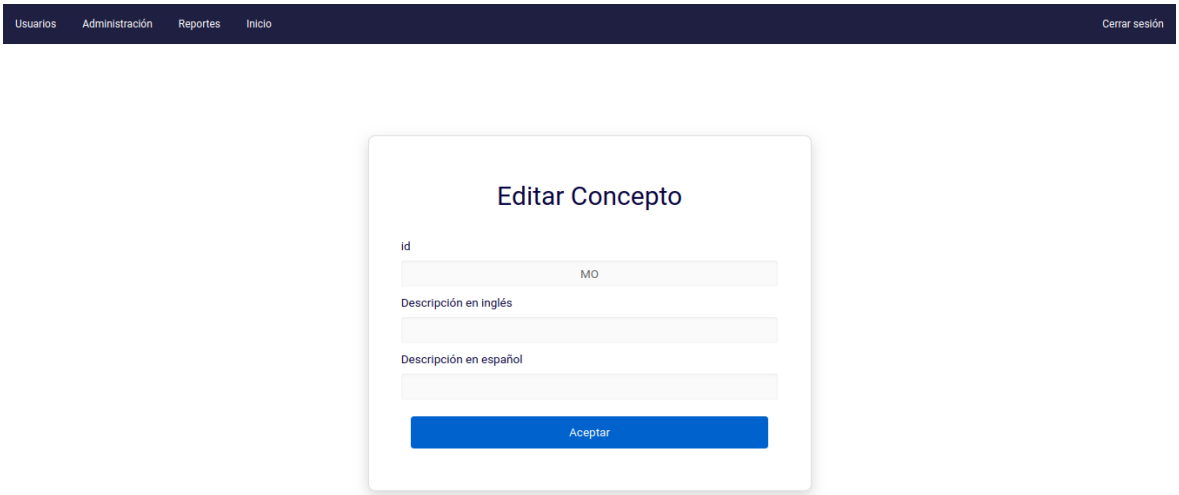

Figura 4.14: Vista para editar los conceptos que se encuentran en el sistema.

- La figura 4.15 muestra la vista de los listados de los reportes guardados.

Figura 4.15: Vista para listar los reportes guardados.

- En la figura 4.16 se observa la vista donde se crean los reportes. Esta vista también se utiliza para editar los reportes guardados.

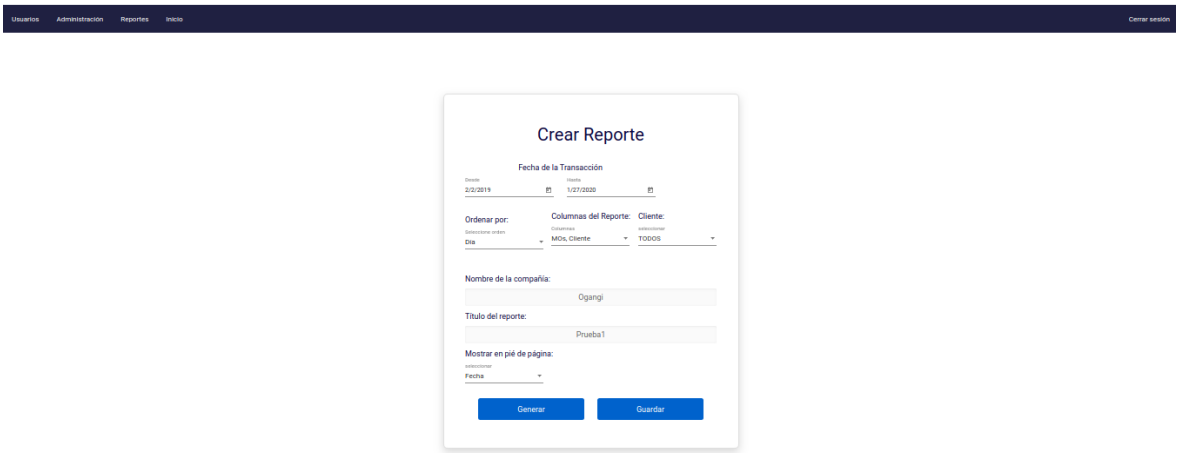

Figura 4.16: Vista para crear reportes.

Capítulo 5

Implementación y pruebas

En este capítulo se describe la manera en cómo se implementaron los módulos del sistema comenzando por la implementación de la base de datos en postgresql, la cual se realizó por medio de una API de persistencia que permite crear la estructura de la base de datos. Luego se explica como se implementó el módulo del procesamiento en paralelo de CDRs nombrando las funciones mas importantes utilizadas para lograr dicho procesamiento, seguido del módulo de generación de reportes y por último la implementación de la interfaz WEB.

5.1. Implementación de la base de datos

Las tablas de la base de datos se implementaron con una API de persistencia de Java(JPA, por sus siglas en inglés). Este es un framework del lenguaje de programación Java que maneja datos relacionales[4] en aplicaciones, permitiendo convertir datos de un lenguaje orientado a objetos a una base de datos relacional.

Cada tabla de esta base de datos es una clase de Java, y por ende, se utilizan anotaciones de JPA para decirle al sistema cuales de estas clases serán una tabla de la base de datos. En el siguiente fragmento de código se aprecian dos anotaciones donde @*Entity* informa al sistema que es una tabla de la base de datos y @*Table* permite definir el nombre que tendrá la tabla en dicha base de datos.

[4]Es un tipo de base de datos que cumple con el modelo relacional

```
@Entity
@Table(name = "sourcefile")
public class SourceFile implements Serializable {
...
}
```

Los campos se definen como variables dentro de la clase y el nombre de cada variable es el nombre de cada columna en la tabla. Esa asignación se puede anular utilizando una anotación llamada @*Column*.

```
@Entity
@Table(name = "sourcefile")
public class SourceFile implements Serializable {
...
@Column(name = "id", columnDefinition = "Decimal(3,0)")
private Short id;

...
}
```


Dentro de la etiqueta @*Column*, se pueden validar los campos de la tabla. En el código anterior, se está diciendo que el campo id de la tabla *sourcefile*, será un entero corto donde sólo aceptará tres números. Como es mapeo de objecto a relacional, es obligatorio utilizar una clave primaria y se define con la anotación @*Id*.

```
@Entity
@Table(name = "sourcefile")
public class SourceFile implements Serializable {

@Id
@Column(name = "id", columnDefinition = "Decimal(3,0)")
private Short id;

@Column(name = "fullname", length = 200, nullable = false)
private String fullname;
```

```
11
12  @Column(name = "post_lastread", columnDefinition = "Decimal(10,0)",
13  nullable = false)
14  private Long post_lastread;
15
16  @Column(name = "active", columnDefinition = "Decimal(1,0)",
17   nullable = false)
18  private Short active;
19  ...
20  }
```

Las relaciones uno a muchos también se crean utilizando las etiquetas llamada *@OneToMany* y *@ManyToOne*. En el código se podrá observar la clase *SourceFile*, que es la tabla que se encarga de guardar las direcciones de los archivos que contienen los datos del procesamiento de tráfico para luego ser procesados por el módulo de procesamiento de CDRs.


```
1   @Entity
2   @Table(name = "sourcefile")
3   public class SourceFile implements Serializable {
4
5   private static final long serialVersionUID = -7113027716996210246L;
6
7   @Id
8   @Column(name = "id", columnDefinition = "Decimal(3,0)")
9   private Short id;
10
11  @Column(name = "fullname", length = 200, nullable = false)
12  private String fullname;
13
14  @Column(name = "post_lastread", columnDefinition = "Decimal(10,0)",
15   nullable = false)
16  private Long post_lastread;
17
18  @Column(name = "active", columnDefinition = "Decimal(1,0)",
19   nullable = false)
```

```
20  private Short active;
21
22  @ManyToOne(fetch = FetchType.LAZY)
23  @JoinColumns(foreignKey = @ForeignKey(name = "fk_sourcefileConcept"),
24  value = {@JoinColumn(name = "conceptid", nullable = false,
25   referencedColumnName = "id")})
26  private Concept sourcefileConcept;
27
28  @ManyToOne
29  @JoinColumns (foreignKey = @ForeignKey(name = "fk_sourcefile_carrier"),
30  value = { @JoinColumn(name = "carrierid", referencedColumnName = "id")})
31  private Carrier sourcefileCarrier;
32
33  @Column(name = "line_lastread", length = 4000)
34  private String line_lastread;
35
36  @Column(name = "clientid", columnDefinition = "Decimal(3,0)")
37  private Short clientid;
38
39  @Column(name = "sc_to_exclude", length = 4000)
40  private String sc_to_exclude;
41
42  @OneToMany(fetch = FetchType.LAZY, mappedBy = "sourcefile_index",
43   cascade = CascadeType.ALL)
44  @JsonManagedReference
45  private Set<SourceFileIndex> index;
46
47  //      Getters and Setters
48  ...
```


Siguiendo la lógica de la figura (4.2) y lo antes mencionado, se logró implementar la base de datos relacional utilizada en este proyecto.

5.2. Implementación del módulo de procesamiento de CDRs

Para implementar el módulo de procesamiento de CDRs se utilizó el framenwork Spring y como lenguaje de programación, Java. Siguiendo el diseño de la figura 4.4, se crearon tres clases. La primera clase tiene por nombre *FilePreprocessing*, esta clase hace una consulta a la base de datos, específicamente a la tabla *sourcefile* para consultar la dirección de los archivos, el concepto del archivo (Mo o MT) y el tipo de archivo que debe procesar. Luego se implementó la clase *CreateThreads* donde se crean los hilos que se utilizan para el procesamiento en paralelo. Para saber cuantos hilos utilizar se implemento una método llamado *java.lang.Runtime.availableProcessors*() el cual devuelve el número máximo de procesadores disponibles para la maquina virtual de Java, esto facilita el uso del sistema en cualquier máquina ya que se obtendrá siempre la cantidad de procesos que se pueden realizar.

Al tener la cantidad de procesadores, se llama al constructor de la clase *FilePreprocessing* para poder utilizar la lista que tiene de los archivos a procesar, estos se dividen en cantidades iguales dependiendo de la cantidad de hilos que se crearán, luego estos archivos se leen de manera secuencial y las líneas se cargan en memoria en diferentes listas (una lista por cada hilo) que son enviadas como parámetro de entrada a cada uno de los hilos. Ejemplo: Si la computadora tiene disponible cuatro procesadores, se dividirán los archivos en cantidades iguales para luego ser procesados y cargadas en memoria en cuatros listas de tipo *ArrayList.class*. Si no se puede dividir la información en tamaños iguales, se crearan tres listas con la misma cantidad de archivos y la última sublista se quedará con el resto de archivos, luego se leen de manera secuencial los archivos y se crean tres listas con la misma cantidad de líneas y la última lista con el resto de las líneas.

Para crear los hilos se implementó la tercera clase llamada *ReadThreads*, esta clase extiende de *Thread* que trae el método *run*() el cual se debe redefinir. Este método es invocado cuando se inicia el hilo mediante una llamada al método *start*(). El hilo se inicia con la llamada al método *run*() y termina con este.

El constructor de la clase *ReadThreads* es llamado dentro de la clase

CreateThreads para crear el hilo. Este está siendo creado dentro de una estructura de repetición seguido de la llamada al método *start*().

```
1
2  public class CreateThreads {
3
4  ...
5  if(tamList < cantThreads) {
6   int countThread = 0;
7   int nameThread = 1;
8
9   for(int i = startFiles; i< intTamList; i++) {
10  //                                    Un archivo para cada hilo
11
12   if((countThread < cantThreads)&&(i< listJson.size())) {
13    List<JsonLines> jsonListLines = new ArrayList<JsonLines>();
14    jsonListLines.add(readLines(listJson.get(i)));
15    ReadThreads thread = new ReadThreads(jsonListLines, token);
16    thread.setName("hilo " + nameThread);
17    thread.start();
18    refThread.add(thread);
19    nameThread++;
20   }
21   countThread++;
22  }
23 }else if(tamList > cantThreads) {
24  for(int i = 1;  i <= cantThreads; i++) {
25   List<JsonLines> jsonListLines = new ArrayList<JsonLines>();
26   for(j = startFiles; j < intFinalFiles; j++) {
27    jsonListLines.add(readLines(listJson.get(j)));
28   }
29   if((i == cantThreads)) {
30    for(int k = intFinalFiles; k < tamList; k++) {
31     jsonListLines.add(readLines(listJson.get(k)));
32    }
33   }
34   ReadThreads thread = new ReadThreads(jsonListLines, token);
```

```
35    thread.setName("hilo " + i);
36    thread.start();
37    refThread.add(thread);
38    startFiles = intFinalFiles;
39    intFinalFiles += cant;
40   }
41 }
42 ...
43 }
```

Dentro del método *run*() se implementa la funcionalidad que utilizará cada hilo. Estos hilos leen cada lista que tienen como parámetro de entrada y procesan cada posición de la lista que contiene una variable de tipo *ArrayList.class* con las líneas a procesar, otra variable contiene el concepto (MO o MT) y una última variable que tiene el tipo de archivo. Antes de procesar la lista con las líneas, se compara si la lista con las líneas es de tipo Mo o MT y luego comienza a leer cada lista y a procesar la línea para obtener la información que se necesita para guardar en la base de datos.

Al leer la línea, se llama un método encargado de procesar la misma, este método por medio de expresiones regulares guarda en variables la información que se necesita de la línea y la ordena en variables dentro de una función llamada *SMPPGWRecord*(). La forma en como se procesa la línea con el método varía dependiendo de si es de tipo de concepto MO o MT ya que la línea procesada no es igual para ambos casos. Luego de tener la información importante de la línea, se llama al método *processRecord*() que tiene como parámetros de entrada la información de la función *SMPPGWRecord*() y el tipo de archivo. Esta clase, consulta con el tipo de archivo si ya existe información en la base de datos, recolecta un valor necesario para guardar con la información ya recolectada de la base de datos, la información se convierte en una línea de tipo carácter para ser guardada en un arreglo. Luego de tener todas las líneas procesadas y en el arreglo, se procede a llamar al método *storeChanges*() que tiene como parámetro de entrada un arreglo de caracteres obtenido en la función anterior, se procede a llamar a un procedimiento almacenado que procesa el arreglo, divide la línea de caracteres y guarda cada división en las variables correspondientes para finalmente insertar la línea en la base de datos agrupada por cada media hora. Ejemplo: Hay cuatro campos de la

línea ya procesada que deben ser iguales, entre esos campos se encuentra la fecha que tiene como formato $YYY - MM - DDHH24 : MM : SS$; si la hora está por debajo de los 30 minutos se agrupa por la hora, si está por encima de los 30 minutos, se agrupa por la hora en la que se encuentre con los treinta minutos. luego se hace una consulta a la tabla *mtmo_hourly* para cerciorar si no existe esa fecha en la base de datos con los otros tres campos extraídos de la línea procesada. Si no existe, se inserta una nueva línea en la tabla, pero si existe, entonces se procede a sumar uno al contador de la línea ya existente en la base de datos y de esta manera se mantiene la información actualizada de cuantos mensajes de tipo MT y MO hay con los cuatro campos de la línea procesada.

5.3. Implementación del módulo para generar reportes automáticos

Para la implementación de este módulo, se creó un método en Spring que recibe como parámetros los datos que se explican en la sección (4.4.) Este método llama a un procedimiento almacenado el cual recibe los mismos parámetros de entrada, selecciona la consulta correspondiente y devuelve una variable de cursor[5] con la respuesta; esta es enviada a la interfaz y se genera el archivo de tipo CSV con la información solicitada.

5.4. Implementación de la interfaz WEB

La interfaz se implementó en angular (ver sección 2.5), para el desarrollo de la misma se utilizaron dos librerías que permitieron desarrollar la aplicación de manera mas rápida. Estas librerías son: Bootstrap, que es un framework de código abierto el cual contiene plantillas de diseño basadas en CSS y en Javascript para formularios, botones, navegación y otros componentes de la interfaz que facilitan el diseño de la misma y Angular material que es una biblioteca de componentes para desarrolladores de Angular; estos componentes ayudan al desarrollo de aplicaciones web facilitando el

[5]Un cursor, encapsula la consulta y luego lee el resultado de la consulta unas pocas filas a la vez, evitando el desbordamiento de memoria cuando el resultado contiene una gran cantidad de filas.

diseño del mismo.

5.5. Pruebas de rendimiento

Para obtener el rendimiento del procesamiento en paralelo se utilizó la fórmula de aceleración propuesta por Gene Amdahl, la cual puede ser definida como [8]

$$S(P, N) = \frac{T(1, N)}{T(P, N)} \tag{5.1}$$

Donde $T(1, N)$ es el tiempo de ejecución secuencial conocido para resolver un problema y $T(P, N)$ es el tiempo de ejecución en un sistema con P procesadores para resolver el mismo problema. Si el resultado es mayor que uno, significa que se mejoró el sistema, en el caso de ser menor que uno significa que empeoró. Si el valor es igual a uno, el sistema no se mejoró ni empeoró utilizando programación paralela.

Para comprobar la aceleración del sistema se realizaron varias pruebas en una computadora que tiene las siguientes características:

- Un procesador Intel(R) Core(TM) i7-2760QM CPU @ 2.40GHz.
- Sockets: 1.
- Núcleos por socket: 4.
- Hilos de procesamiento por núcleo: 2.

En la tabla (5.1) están registrados los diferentes escenarios que se utilizaron para poder calcular la aceleración de la aplicación tomando el tiempo en secuencial del sistema actual y el tiempo en paralelo del sistema nuevo con el módulo de procesamiento en paralelo.

Aplicando la fórmula de la aceleración de Amdahl se consiguieron los siguientes resultados

	Archivos	Cantidad de lineas	Cantidad de hilos	Tiempo en milisegundos	Tiempo en minutos
Prueba 1.	30	todos de 10.000 lineas.			
Paralelo.			8	3295096	54.91826667
Secuencial.			1	3415568	56.92613333
Prueba 2.	30	Lineas hasta 10.000			
Paralelo.			8	122373	2.03955
Secuencial.			1	826638	13.7773
Prueba 3.	30	Lineas hasta 10.000			
Paralelo.			8	122547	2.04245
Secuencial.			1	909166	15.1527667
Prueba 4.	30	Lineas hasta 30.000			
Paralelo.			8	1434187	23.90311667
Paralelo.			6	1633807	27.23011667
Secuencial.			1	3051932	50.86553333
Prueba 5.	16	todos de 37.500			
Paralelo.			8	3167836	52.79726667
Secuencial.			1	6712295	111.8715833

Tabla 5.1: Pruebas del sistema del procesamiento en paralelo del CDRManager

	Tiempo en milisegundos del procesamiento secuencial	Tiempo en milisegundos del procesamiento en paralelo	Resultado
Prueba 1.	3415568	3295096	1,036560999
Prueba 2.	826638	122373	6,75506852
Prueba 3.	909166	122547	7,418916824
Prueba 4.	3051932	1434187(Con 8 hilos)	2,127987494
Prueba 4.	3051932	1633807(Con 6 hilos)	1,867988079
Prueba 5.	6712295	3167836	2,118889677

Tabla 5.2: Resultados obtenidos con la ecuación de Amdalh para la aceleración.

En la tabla (5.3) Se observa que el sistema en paralelo mejoró significativamente, cumpliendo con el objetivo de este trabajo de grado.

	Resultado en milisegundos	Porcentaje de mejoría
Prueba 1.	1,036560999	96,5
Prueba 2.	6,75506852	14,8
Prueba 3.	7,418916824	13,5
Prueba 4.	2,127987494	46,9 (Con 8 hilos)
Prueba 4.	1,867988079	53,5 (Con 6 hilos)
Prueba 5.	2,118889677	47,2

Tabla 5.3: Porcentaje de mejoría del sistema.

5.6. Pruebas Unitarias

Las pruebas unitarias de la aplicación implementada en Angular, se realizaron utilizando Jasmine, este es un framework de pruebas de código abierto para Javascript. Este framework viene con un corredor de pruebas llamado Karma.

Estas pruebas se realizaron por componentes, donde cada componente tiene varias funcionalidades de la aplicación. Las pruebas unitarias fueron realizadas a la mayoría de los componentes del sistema. A continuación se muestran los resultados obtenidos de cada uno de los componentes:

File		Statements		Branches		Functions		Lines	
src		100%	3/3	100%	0/0	100%	0/0	100%	3/3
src/app		100%	2/2	100%	0/0	100%	0/0	100%	1/1
src/app/create-reports		75.4%	95/126	53.57%	15/28	72%	18/25	73.28%	85/116
src/app/edit-concept		100%	17/17	100%	0/0	100%	5/5	100%	14/14

Figura 5.1: Resultados. Pruebas unitarias.

En la figura (5.1) se encuentra el componente create-reports, este componente tiene las funcionalidades de crear reportes, editar reportes y eliminar reportes. esta prueba tiene un resultado del 75.4 % a pesar de que las pruebas de las funcionalidades ya mencionadas fueron satisfactorias. Esto se debe a que faltó estructurar de mejor manera

el flujo de ejecución del componente. Luego se observa al componente edit-concept, este contiene la funcionalidad para editar los tipos de mensajes que maneja la aplicación.

src/app/edit-user		98.08%	102/104	85.71%	36/42	100%	26/26	97.94%	95/97
src/app/insert-concept		100%	15/15	100%	0/0	100%	5/5	100%	12/12

Figura 5.2: Resultados. Pruebas unitarias.

En (5.2) se observa el componente edit-user, el cual permite editar los usuarios registrados en el sistema y el componente insert-concept, donde se encuentra la funcionalidad de crear un nuevo tipo de mensaje para el sistema.

src/app/insert-user		100%	53/53	93.33%	14/15	100%	15/15	100%	48/48
src/app/list-concept		100%	14/14	100%	0/0	100%	9/9	100%	11/11

Figura 5.3: Resultados. Pruebas unitarias.

En la figura (5.3) se encuentra el componente insert-user, este tiene la funcionalidad de agregar usuarios al sistema. También se puede observar el componente list-concept, en este componente se encuentra la funcionalidad para ver todos los tipos de mensaje registrados en el sistema, como también la funcionalidad de eliminar los mensajes del sistema.

src/app/login		100%	22/22	50%	1/2	100%	7/7	100%	19/19
src/app/manual-usuario		100%	4/4	100%	0/0	100%	2/2	100%	3/3

Figura 5.4: Resultados. Pruebas unitarias.

En la figura (5.4) se observan los componentes del inicio de sesión del sistema y del manual de usuario donde se explica la forma en como se debe crear un reporte.

src/app/menu-admin		81.25%	26/32	33.33%	4/12	75%	6/8	79.31%	23/29
src/app/models		100%	6/6	100%	0/0	100%	6/6	100%	6/6
src/app/report-users		100%	7/7	100%	0/0	100%	3/3	100%	5/5

Figura 5.5: Resultados. Pruebas unitarias.

En la figura (5.5) se encuentra el componente del menú para el administrador, el componente report-users que muestra los tipos de reportes que se pueden generar, en

este caso solo de mensajes de texto y el componente models que contiene estructuras utilizadas para la implementación del sistema.

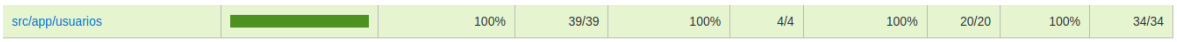

Figura 5.6: Resultados. Pruebas unitarias.

La figura (5.6) muestra el resultado del componente usuarios, este componente tiene la funcionalidad de mostrar la lista de los usuarios que están registrados en el sistema.

De esta manera se puede observar que se obtuvo un resultado satisfactorio en las pruebas unitarias para la amyoría de las funcionalidades del sistema CDRManager.

Capítulo 6

Conclusiones

En este proyecto de grado se desarrolló un sistema multihilo para el procesamiento de tráfico y un sistema para la generación de reportes para la compañía Ogangi de Venezuela C.A. con el objetivo de mejorar los tiempos de procesamiento del tráfico de datos de la compañía y generar los reportes de una manera sencilla.

Para lograr esto, se realizó una investigación para establecer los criterios necesarios en el diseño e implementación de los módulos, logrando de manera exitosa realizar el diseño del módulo que se encargará de procesar los CDRs, el módulo que se encargará de generar reportes automáticos, la base de datos y la interfaz WEB para la generación de reportes seguido de la implementación del módulo que se encargará de procesar los CDRs, el módulo que se encargará de generar reportes automáticos, la base de datos y la interfaz WEB para la generación de reportes

Capítulo 7

Recomendaciones y trabajos futuros

7.1. Recomendaciones

Se recomienda realizar pruebas de procesamiento del tráfico de los CDRs en el servidor en el que se instalará el nuevo sistema para seleccionar el número máximo de archivos a consultar en cada procesamiento y el número de lineas máxima que se podrán procesar de cada archivo para evitar que el servidor se quede sin memoria y falle en el procesamiento del tráfico de datos.

7.2. Trabajos futuros

Como trabajo futuro, se implementará en el sistema CDRManager poder procesar lineas de mensajes Delivery Receipt. Este mensaje es enviado por una operadora identificando si el mensaje MT o MO ha sido recibido o rechazado.

Bibliografía

[1] "About postgresql," The PostgreSQL Global Development Group (1996-2019), PostgreSQL, https://www.postgresql.org/about/.

[2] "Architecture overview," Google (2010-2019), Angular, https://spring.io/projects/spring-framework.

[3] "Class thread," sin fecha, Java™ Platform Standard Ed. 7, https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html.

[4] "Qué es scrum," sin fecha, proyectos agiles.org, https://proyectosagiles.org/que-es-scrum/.

[5] "Spring framework," (2019), Spring by Pivotal, https://spring.io/projects/spring-framework.

[6] "What is java technology and why do i need it?" sin fecha, Java™, https://www.java.com/en/download/faq/whatis_java.xml.

[7] "¿cómo funciona la metodología scrum," sin fecha, Platzi, https://platzi.com/blog/metodologia-scrum-fases/.

[8] J. Aguilar and E. Leiss, *Introducción a la Computación Paralela*, 2004.

[9] S. Ekanayake, S. Kamburugamuve, P. Wickramasinghe, and G. C. Fox, "Java thread and process performance for parallel machine learning on multicore hpc clusters," *in Big Data (Big Data)*, pp. 347–354, 2016.

[10] T. Mattson, B. Sanders, and B. Massingill, *Patterns for Parallel Programming*, 2005.

[11] M. Mrva, K. Buchenrieder, and R. Kress, "A scalable architecture for multi-threaded java applications," *IEEE*, pp. 868–874, 1998.

[12] S. Shakya, R. S. Chaulagain, S. Pandey, and P. Gyawali, "Distributed high performance computing using java," *IEEE*, 2017.

[13] A. Silberschatz, *Fundamentos de sistemas operativos, 7ma Edición*, 2005.

[14] G. L. Taboada, "Design of Efficient Java Communications for High Performance Computing," Ph.D. dissertation, Department of Electronics and Systems University of A Coruña, Spain, May 2009.

[15] A. Tanenbaum, *Sistemas operativos modernos, 3ra Edición*, 2009.

[16] A. Vargas, L. Garcia, S. Martinez, L. Chavez, and D. Munoz, "Cifrado de datos con algoritmo aes usando programación multihilo en java," *Instituto tecnológico de CD. Victoria, Tecnológico nacional de Mexico*, 2010.

[17] G. Wolfmann, "Análisis de patrones de paralelismo bajo la óptica de las aplicaciones de cómputo científico sobre clusters de nodos multicore," Master's thesis, Facultad de Informática. Universidad Nacional de La Plata, diciembre 2010.