UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

Desarrollo de plataforma de procesamiento paralelo distribuido para la ejecución de diferentes trabajos

Este Jurado; una vez realizado el examen del presente trabajo ha evaluado su contenido con el resultado: Diez y ocho (18)

J U R A D O E X A M I N A D O R

Firma:	Firma:	Firma:
Nombre: Wilmer Pereira	Nombre: Luis Avendaño	Nombre: Saúl Buitrago

REALIZADO POR	Díaz Conde, Gabriel Jesús Salicetti Benellán, Giselle Carolina
PROFESOR GUÍA	Buitrago, Saúl
FECHA	Caracas Julio de 2006.

ÍNDICE GENERAL

APÉNDICES

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICOS

SINOPSIS

El presente trabajo especial de grado consiste en el diseño y desarrollo de una plataforma de procesamiento paralelo distribuido para la ejecución de diferentes trabajos sobre la arquitectura de un cluster. Dicha plataforma cuenta con un conjunto de procesos cada uno de ellos con tareas muy específicas, entre las cuales se pueden citar: autenticación de usuarios, verificación de disponibilidad de máquinas, distribución de tareas en máquinas asignadas, monitoreo constante de tareas y recuperación de fallas. Como medio de comunicación entre procesos se empleará como librería de paso de mensajes la PVM o Parallel Virtual Machine, que es una herramienta capaz de crear una máquina virtual paralela empleando para ello los recursos libres de cada uno de los nodos, haciendo creer que se está ante una única supermáquina.

La metodología empleada es el Prototipo Evolutivo, con la cual es posible ofrecer muestras tangibles del desarrollo del proyecto a través de prototipos funcionales, de los cuales se obtiene una retroalimentación del cliente que da la posibilidad de adaptar la aplicación según las sugerencias recibidas.

Como resultado del desarrollo se obtuvo un Cluster Beowulf y una Plataforma de Procesamiento Paralelo Distribuido (APPD), conformada por seis procesos que le permiten llevar a cabo sus labores:

Proceso Inicio Plataforma, encargado de dar comienzo el ambiente APPD, de permanecer a la espera de mensajes provenientes del resto de los procesos y de iniciar al Proceso Inicializador Servidor, que a su vez tiene como objetivo recibir las solicitudes de servicios provenientes del proceso Inicializador Cliente. Para proveer dichos servicios activa al Proceso Central, que verifica la disponibilidad de máquinas y realiza la autenticación de usuarios. Finalmente el Proceso Maestro debe distribuir las tareas a los Procesos Esclavos y permanecer en constante monitoreo de las ejecuciones, a fin de detectar tareas culminadas y fallas ocurridas, para así proveer un mecanismo de recuperación.

Los procesos antes mencionados fueron implementados con el lenguaje de programación C, el cual ofrece economía sintáctica, control de flujo, estructuras sencillas y un buen conjunto de operadores. Como manejador de bases de datos se empleó MySQL por ser de libre distribución y de fácil

adquisición. Para el montaje del cluster Beowulf fue necesaria la instalación del sistema operativo Fedora Core 4 en todas las máquinas, además de la configuración del protocolo NFS o Network File System, encargado de compartir carpetas de forma tal que puedan ser accedidas remotamente. Finalmente para la administración de usuarios se empleó NIS o Network Information System, cuya finalidad es crear una base de datos centralizada de los usuarios que tienen acceso a las máquinas que conforman el cluster.

INTRODUCCIÓN

En el mundo de hoy en día se ha experimentado un creciente interés por el procesamiento paralelo distribuido en lo que respecta a la ciencia de la computación. Esto se debe a un incremento en la necesidad de potencia a la hora de realizar cómputos para reducir el tiempo de procesamiento en problemas de cálculo intensivos o de grandes volúmenes de datos, donde los requerimientos en relación al tiempo de respuesta son críticos.

Este tipo de arquitecturas ofrece la posibilidad de distribuir el procesamiento en grandes redes de computadores que pueden o no ser homogéneos, minimizando así el tiempo total de ejecución. También pueden ser empleadas en sistemas en los que no es tan importante la velocidad de cómputo sino la necesidad de resolver problemas en más de una ubicación física a la vez, capacidad que puede asociarse a una configuración paralela, surgiendo así el concepto de trabajo cooperativo, el cual es realizado a través de módulos o funciones que llevan a cabo tareas individuales, pero que a su vez cooperan entre si para lograr un objetivo común. Estos módulos pueden ser ejecutados en paralelo en una o más computadoras o procesadores, obteniendo como beneficio un incremento en la velocidad de procesamiento, además de facilitar la cooperación misma entre ellos. Dicha cooperación suele estar vinculada al intercambio de información, esto puede ser posible utilizando espacios de almacenamiento común, al que todas las funciones involucradas tengan acceso de lectura y escritura o a través de librerías de paso de mensajes. Éste último paradigma ha cobrado mayor relevancia con el paso del tiempo debido a las virtudes que ofrece, como por ejemplo hacer uso de equipos existentes conectados a través de una red, utilizando el poder de cómputo excedente de cada uno de ellos.

Una forma eficiente de aprovechar estos equipos existentes es a través de los clusters, los cuales están conformados por un conjunto de computadoras interconectadas con dispositivos de alta velocidad que actúan en conjunto usando el poder cómputo de varios CPU en combinación para resolver ciertos problemas dados.

Basados en lo anteriormente expuesto es importante un medio que permita el cómputo paralelo / distribuido, para montar sobre dicho cluster aplicaciones cooperativas permitiendo su ejecución de forma eficiente, ofreciendo transparencia al usuario puesto que no deberá preocuparse por la heterogeneidad de hardware y software de las máquinas en las que se trabaja, ni tampoco en lo que

se refiere a los medios de comunicación utilizados para la transmisión de información, siendo posible que las computadoras pueden ser visualizadas y utilizadas como un solo recurso computacional que actúa como un supercomputador virtual paralelo.

La metodología empleada para el desarrollo es el Prototipo Evolutivo, en la cual se van elaborando versiones de la aplicación que son mostradas al cliente y se refina el producto en función de la retroalimentación del mismo. Con esta metodología es posible mostrar signos visibles de progreso permitiendo además generar un sistema altamente fiable.

Este documento pretende describir el desarrollo del cluster y de los distintos procesos que conforman la aplicación de procesamiento paralelo distribuido, siguiendo el siguiente esquema:

- **Capítulo I:** En éste capitulo se expone el planteamiento del problema, objetivos general y específicos del trabajo especial de grado, su justificación, alcance y limitaciones tanto del cluster como de la plataforma desarrollada.
- **Capítulo II:** Este capítulo muestra toda la investigación teórica realizada en la cual se basó la realización de este trabajo especial de grado.
- **Capítulo III:** Descripción de la metodología empleada para el desarrollo de la plataforma y la justificación de su elección.
- **Capítulo IV:** Explica de forma detallada como se completaron cada una de las etapas de la metodología para el cumplimiento de los objetivos planteados.
- **Capítulo V:** Expone los resultados obtenidos al aplicar la metodología seleccionada para la elaboración del presente trabajo de grado.

Finalmente, se muestran las conclusiones, recomendaciones y bibliografía de los libros consultados.

CAPÍTULO I
PLANTEAMIENTO DEL PROBLEMA

I.1 Planteamiento del problema

La velocidad se ha convertido en una de las principales demandas en el área de los sistemas computacionales, debido a que facilitan o agilizan el trabajo para diferentes tipos de investigaciones, cálculo numérico, simulación, modelaje numérico de problemas científicos, procesamiento de imágenes y gran variedad de aplicaciones de ingeniería.

Para realizar este tipo de trabajos es necesario el procesamiento de grandes cantidades de datos, numerosos y repetitivos cálculos numéricos, lo cual conduce al empleo de una cantidad de tiempo considerable en dichos procesamientos. Es por esto que han sido creadas las denominadas Supercomputadoras, las cuales poseen características especiales en cuanto a hardware que permiten el procesamiento y cómputo de información a grandes velocidades y en muy corto tiempo. Sin embargo una desventaja de este tipo de computadoras es que resultan extremadamente costosas de adquirir y de mantener actualizadas tanto en la plataforma de software como de hardware.

Es por esto que se plantea la simulación de un Supercomputador a través del desarrollo de una plataforma para procesamiento paralelo distribuido, para lo cual se requerirá la construcción de un cluster conformado por un conjunto de computadores personales que estarán interconectadas a través de una red y actuarán como una sola máquina con la capacidad, velocidad y software necesarios para abordar este tipo de problemas complejos, resolverlos en corto tiempo y de forma eficaz.

I.2 Objetivos

I.2.1 Objetivo General

Desarrollar una plataforma para procesamiento paralelo distribuido que permita distribuir data, parámetros o información y ejecutar un conjunto de trabajos relacionados entre si y posiblemente distintos, sobre un cluster.

I.2.2 Objetivos Específicos

- Elaborar una investigación de base teórica acerca de algunas de las librerías de comunicación disponibles para plataformas paralelas y distribuidas.

En base a la investigación teórica realizada:

- Ensamblar y configurar el cluster: varias estaciones de trabajo/PCs , sobre las cuáles se instalará el sistema operativo LINUX , así como las librerías de pase de mensajes previamente investigadas.
- Realizar pruebas del funcionamiento del cluster con ejemplos sencillos.
- Diseñar, desarrollar e implantar el ambiente de procesamiento paralelo distribuido APPD (para la ejecución distribuida de aplicaciones sobre el cluster), el cual permitirá controlar el acceso de los usuarios a la herramienta, distribuir entre las máquinas disponibles la data, parámetros o información necesaria para la ejecución de los procesos previamente instalados, devolver resultados, reasignar la data, parámetros o información en caso de falla de la máquina donde estaban corriendo y finalizar o terminar una ejecución.
- Evaluar la plataforma mediante el uso de una aplicación con alto potencial para el procesamiento paralelo distribuido.

I.3 Justificación

Para el mundo de hoy en día resulta de gran importancia la utilización de computadores que permitan procesar la información y realizar grandes cómputos que contribuyan en diferentes investigaciones.

Es por esto que una plataforma con estas características será de gran importancia para la Universidad Católica Andrés Bello, puesto que permitirá impulsar, sin lugar a dudas, la producción de conocimiento científico en áreas como cálculo numérico, simulación y modelaje, computación grafica, física computacional, química computacional, procesamiento masivo paralelo, procesamiento paralelo distribuido, bases de datos paralelas y sistemas de memoria distribuida, entre otras.

En lo académico, este tipo de arquitectura, apoyará el desarrollo de asignaturas relacionadas con Programación Paralela y Distribuida, Arquitectura de Computadoras, Sistemas de Tiempo Real y Redes de Computadores. Por otro lado, el campo del cómputo paralelo y distribuido ha experimentado un gran avance en los últimos años, estos avances deben ser introducidos en la universidad, para ofrecer a los estudiantes un panorama actualizado de las diferentes arquitecturas usadas en la actualidad por la industria y academia.

Por todo lo anteriormente mencionado es que se propone la realización de un ambiente de procesamiento paralelo distribuido apoyado en el hardware y software necesarios, a fin de que permita impulsar el desarrollo académico en la universidad, y servirá de base para el desarrollo de nuevas funcionalidades que hagan que esta herramienta sea cada día más útil y poderosa.

I.4 Limitaciones y Alcance

I.4.1 Limitaciones

- El usuario podrá especificar el número de máquinas que necesita para su ejecución o se asignará un conjunto de máquinas disponibles por defecto.
- La plataforma será utilizada mediante la ejecución de códigos precompilados en lenguaje C o Fortran, que serán desarrollados o codificados y ejecutados por el usuario.
- La herramienta podrá ser utilizada por más de un usuario a la vez.
- Los procedimientos o programas a ejecutar en paralelo a fin de probar la plataforma serán previamente instalados en las máquinas disponibles. Estos programas o procedimientos son facilitados por el usuario, es decir, ya se encuentran codificados.
- Las Entradas de la plataforma será la data, parámetros o información a ser utilizada por los programas o procedimientos antes mencionados.
- Las Salidas, serán los resultados de la ejecución de los procedimientos a través de vectores o archivos de textos.

I.4.2 Alcance

La aplicación de procesamiento paralelo distribuido a desarrollar estará conformada por los siguientes componentes o procesos:

- Proceso inicializador, encargado de iniciar el proceso central. Una vez iniciado el proceso central se mantendrá en espera de mensajes de la plataforma, referentes a fallas, disponibilidad de máquinas y procesos culminados.
- Proceso central, encargado de controlar el acceso de los usuarios autorizados a la plataforma (autenticación de usuarios) y de administrar las máquinas disponibles.
- Proceso maestro, que permitirá a los usuarios ejecutar las siguientes tareas: distribuir entre las máquinas disponibles la data, parámetros o información necesaria para la ejecución de los procedimientos previamente instalados, devolver resultados, reasignar la data, parámetros o información en caso de falla de la de la máquina donde estaban corriendo (recuperación de fallas provenientes de la red, de la máquina o del proceso) y finalizar o terminar una ejecución.
- Proceso esclavo, estará ubicado en cada una de las máquinas que serán utilizadas para la ejecución de los procedimientos, está encargado de recibir la información, data o parámetros provenientes del proceso maestro y ejecutar el procedimiento con dicha data.

CAPÍTULO II
MARCO TEÓRICO

Este capítulo contiene un marco teórico que presenta algunos de los conocimientos necesarios para la comprensión del documento. A lo largo del capítulo se exponen los conceptos y tecnologías utilizadas en la construcción del sistema.

II.1 Cluster:

Un Cluster es un conjunto de computadoras interconectadas con dispositivos de alta velocidad que actúan cooperativamente usando el poder de cómputo de varios CPU en combinación para resolver ciertos problemas dados. Estos equipos trabajan de forma independiente ejecutando las aplicaciones de forma conjuntan y aparecen ante clientes y aplicaciones como un solo sistema. Los clusters permiten aumentar la escalabilidad, disponibilidad y fiabilidad de múltiples niveles de red.

La escalabilidad es la capacidad de un equipo para hacer frente a volúmenes de trabajo cada vez mayores sin, por ello, dejar de prestar un nivel de rendimiento aceptable. Existen dos tipos de escalabilidad:

- Escalabilidad del hardware (también denominada escalamiento vertical). Se basa en la utilización de un gran equipo cuya capacidad se aumenta a medida que lo exige la carga de trabajo existente.

- Escalabilidad del software (también denominada escalamiento horizontal). Se basa, en la utilización de un cluster compuesto de varios equipos de mediana potencia. Se utilizan el término RAC (Redundant Array of Computers o Array redundante de equipos) para referirse a los clusters de escalamiento horizontal. Del mismo modo, es posible añadir nodos a un cluster para aumentar también su rendimiento.

Para mayor información de los clusters consultar apéndice A.

II.1.1 Cluster Beowulf

Cluster Beowulf es una tecnología para agrupar computadores basados en el sistema operativo Linux para formar un supercomputador virtual paralelo. En 1994 bajo el patrocinio del proyecto ESS

del Centro de la Excelencia en Ciencias de los Datos y de la Información del Espacio (CESDIS), Thomas Sterling y Don Becker crearon el primer cluster Beowulf con fines de investigación.

Beowulf posee una arquitectura basada en multicomputadores el cual puede ser utilizado para la computación paralela. Este sistema consiste de un nodo maestro y uno o más nodos esclavos conectados a través de una red Ethernet u otra topología de red. Esta construido con componentes hardware comunes en el mercado, similar a cualquier PC capaz de ejecutar Linux, adaptadores de Ethernet y switches estándares. Como no contiene elementos especiales, es totalmente reproducible.

El nodo maestro controla el cluster entero y presta servicios de sistemas de archivos a los nodos esclavos. Es también la consola del cluster y la conexión hacia el exterior. Las máquinas grandes de Beowulf pueden tener más de un nodo maestro, y otros nodos dedicados a diversas tareas específicas, como por ejemplo, consolas o estaciones de supervisión. En la mayoría de los casos los nodos esclavos de un sistema Beowulf son estaciones simples. Los nodos son configurados y controlados por el nodo maestro, y hacen solamente lo que éste le indique. En una configuración de esclavos sin disco duro, estos incluso no saben su dirección IP hasta que el maestro les dice cuál es.

La topología de red recomendada es un Bus, debido a la facilidad para proporcionar escalabilidad a la hora de agregar nuevos nodos al cluster. Protocolos como Ethernet, Fast Ethernet, GigaEthernet, 10/100 Mbps Switched Ethernet, etc, son tecnologías apropiadas para ser utilizadas en Beowulf.

Beowulf utiliza como sistema operativo cualquier distribución Linux. Además usa bibliotecas de paso de mensajes como PVM y MPI. Se apoyan en herramientas, protocolos o servicios que les permiten gestionar y administrar tanto ficheros y permisologías de acceso, como bases de datos centralizadas de nombres de usuarios y contraseñas de las personas autorizadas a utilizar las máquinas del cluster. A continuación se explican las dos herramientas más usadas:

II.1.2 NFS (Network File System)

Es un sistema distribuido que provee acceso transparente a discos remotos, el cuál permite centralizar la administración de los discos, en lugar de hacer réplicas en cada sistema. Tiene como principal objetivo otorgar permisos de lectura y escritura a las máquinas deseadas en los ficheros seleccionados.

Tiene una arquitectura cliente servidor: El servidor posee uno o varios sistemas de archivos y hace que estén disponibles para los integrantes autorizados de la red y los clientes montan los archivos de uno o varios servidores.

II.1.3 NIS (Network Information System)

Es un sistema de base de datos distribuida que reemplaza archivos de configuración duplicados en cada máquina, por un manejo centralizado de los mismos. Su principal objetivo es almacenar en la máquina servidor la información de los usuarios o grupos de usuarios que pueden tener acceso a cualquier máquina del cluster, permitiendo así administrarlos de forma centralizada lo cual facilita la gestión al incorporar o eliminar usuarios haciéndolo en la base de datos ubicada en el servidor, sin necesidad de realizar los cambios en cada uno de los hosts del cluster.

II.2 Sistemas Distribuidos

A lo largo del siglo XX se experimentaron enormes mejoras en la tecnología de cómputo, desde el surgimiento de las redes de área local (LAN) que permiten interconectar cientos de computadores que comparten datos a altas velocidades, hasta las redes de área amplia (WAN) que conectan a millones de computadores alrededor del mundo. Es por esto que hoy en día es posible reunir un gran número de computadores conectados a través de una red de alta velocidad, que trabajen como un sistema de cómputo o sistema Distribuido.

Un sistema distribuido es una colección de computadoras independientes que aparecen ante los usuarios del sistema como una única computadora. (Andrew Tanenbaum, 1996)

Los sistemas distribuidos están compuestos por recursos informáticos tanto físicos como lógicos, que actúan de forma autónoma pero trabajan coordinadamente y pueden estar asignados a un grupo de tareas en concreto, dichos recursos están distribuidos físicamente y funcionan gracias a una red de comunicaciones. Estos sistemas se caracterizan por ser transparentes, es decir, su funcionamiento es exactamente igual desde todos los puntos de la red e independiente de la ubicación del usuario, es el sistema operativo el encargado de ocultar la naturaleza distribuida del sistema y presentarla como si se tratara de un solo equipo.

Para conocer algunos tipos de arquitecturas de los sistemas distribuidos consultar el apéndice B.

II.3 Procesamiento Paralelo

Los procesadores paralelos son sistemas de computación que están formados por varias máquinas conectadas a través de una red, que cuenta con el software necesario para que las unidades procesen la data conjuntamente. Las unidades de procesamiento pueden comunicarse entre si gracias al uso de memoria compartida o con métodos de paso de mensajes.

II.3.1 Taxonomía de Flynn para arquitectura de computadores

Esta taxonomía fue definida por Flynn en 1966, está basada en el flujo de información dentro de un procesador, clasificado en dos tipos: flujo de instrucciones y flujo de datos. Las arquitecturas del computador están clasificadas en:

II.3.1.1 Arquitectura SIMD (Single Instruction – Multiple Data Streams)

Consta de un computador (front - end) y un arreglo de procesadores que son un conjunto de elementos sincronizados, capaces de ejecutar simultáneamente la misma operación con diferente data. Cada procesador del arreglo posee una pequeña porción de la memoria local donde reside la data mientras es procesada en paralelo. El arreglo de procesadores está conectado al bus de memoria del front – end de manera que éste pueda acceder aleatoriamente a la memoria local de los procesadores como si se tratase de otra memoria.

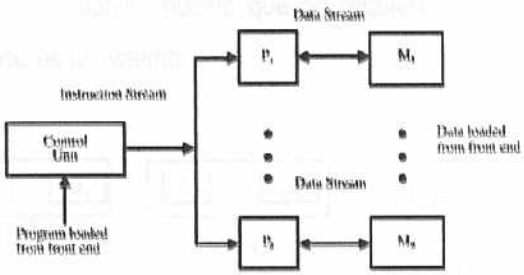

Figura 2.1 Arquitectura SIMD
Fuente: Hesham, Mostafa, (2005).

II.3.1.2 Arquitectura MIMD (Multiple Instructions – Multiple Data Streams)

Este tipo de arquitecturas están conformadas por múltiples procesadores con múltiples módulos de memoria interconectados a través de una red. Se dividen según dos categorías: de memoria compartida o de paso de mensajes.

Los sistemas de memoria compartida implican coordinación entre los procesadores a través de una memoria global que comparten todos los procesadores. Suelen ser servidores que se comunican a través de un bus o memoria caché. Requieren de memorias multipuertos que suelen ser costosas además de necesitar la incorporación del paso de mensajes a la hora de desarrollar las aplicaciones.

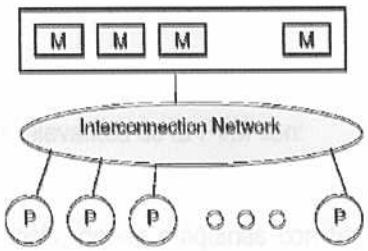

Figura 2.2 Arquitectura MIMD – Memoria Compartida

Fuente: Hesham, Mostafa, (2005).

Los sistemas de paso de mensajes cuentan con la memoria local y el procesador de cada uno de los nodos que se encuentran interconectados a través de una red. No existe memoria global por lo cual es necesario mover la data de una memoria a otra a través del paso de mensajes. Cada procesador tiene acceso a su memoria local, pudiendo almacenar en un buffer los mensajes, usando para esto un espacio de memoria temporal con el fin de esperar a ser enviados o recibidos. Este tipo de sistemas son escalables puesto que no requieren cambios mayores a la hora de incorporar nuevos procesadores al sistema.

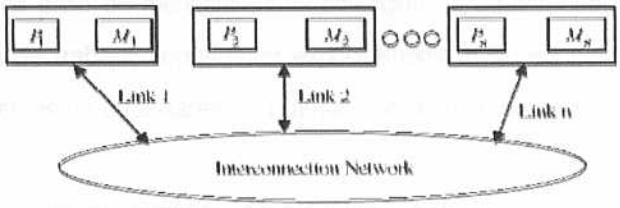

Figura 2.3 Arquitectura MIMD – Paso de mensajes

Fuente: Hesham, Mostafa, (2005).

II.4 Parallel Virtual Machine (PVM)

La PVM es un API GNU de programación C capaz de crear una máquina virtual paralela, es decir, emplea los recursos libres de cada uno de los nodos sin tener que tomar en cuenta el cómo, haciendo creer que se está ante una única supermáquina. Los nodos de la red son denominados hosts, los cuales pueden tener uno o varios procesadores o simplemente funcionar sobre la arquitectura de un cluster. Posee además dos componentes, una librería de rutinas (libpvm) y un demonio (pvmd) que debe residir en cada uno de los hosts de la máquina virtual, el cual sirve como enrutador y controlador de mensajes. Provee un punto de contacto, de autenticación, control de procesos y recuperación de fallas. El demonio de la PVM puede recuperarse de la pérdida de cualquiera de las máquinas excepto de la pérdida del maestro. Si un esclavo pierde a su maestro, él mismo tumba su demonio.

Algunas de las características más relevantes de la PVM son:

- Trabaja con un conjunto de hosts, donde máquinas con uno o con varios procesadores, de memoria compartida o distribuida, pueden formar parte de la misma, siendo posible alterarla agregando o eliminando máquinas durante las operaciones que se realicen en la PVM, lo cual resulta de gran importancia en la tolerancia a fallos.
- Los programas de la PVM pueden ver el ambiente de hardware como una colección de elementos de procesamiento virtual sin atributos o por el contrario pueden elegir explotar las capacidades de una máquina en específico, asignando ciertas tareas a la computadora más adecuada.
- Computación basada en procesos: La unidad de paralelismo en la PVM es una tarea, un hilo de control independiente y secuencial que alterna entre comunicación y cómputo. Generalmente múltiples tareas son ejecutadas en un mismo procesador.
- Modelo explícito de paso de mensajes: Una colección de tareas, cada una procesando una parte de la carga de trabajo, cooperando explícitamente a través del envío y recepción de mensajes provenientes de otras tareas. El tamaño de estos mensajes depende de la memoria disponible.
- Soporta heterogeneidad en términos de máquinas, redes y aplicaciones. Con respecto al mensaje que se envía, la PVM permite que estos contengan más de un tipo de datos, de forma que puedan ser intercambiados con máquinas con diferentes representaciones de data.

Para mayor información de la PVM consultar apéndice C.

II.5 Message Passing Interface (MPI)

Es una interfaz estandarizada para la elaboración de aplicaciones paralelas basadas en el paso de mensajes. El modelo de programación que usa MPI es MIMD (Multiple Instruction - Multiple Data) aunque es posible también utilizar el modelo SPMD (Single Program - Multiple Data). Por ser una interfaz, MPI no exige una determinada implementación del estándar, sino que ofrece una colección de funciones que dan libertad al programador de diseñar su aplicación, sin necesidad de conocer el hardware en el que se va a ejecutar. Tiene como principales objetivos la funcionalidad, la eficiencia y la portabilidad. Incluye paso de mensajes punto a punto y operaciones colectivas, todas ellas dentro del ámbito de un grupo de procesos especificados por el usuario.

Entre las características más relevantes de MPI se encuentran:

- Es una librería para escribir programas.
- Es modular, lo cual agiliza la elaboración de librerías paralelas portables.
- Es extensible de tal manera que se puedan atender a futuras necesidades y desarrollos.
- Permite cómputo heterogéneo, es decir, diferentes tipos de data.
- Ofrece operaciones colectivas, son ejecutadas por todos los procesos que intervienen en el cálculo o comunicación.
- Dispone de un conjunto de tipos de datos amplio permitiendo definir tipos de datos derivados, es decir, datos que se construyen a partir de otros ya existentes.

II.5.1 Comunicación punto a punto

El mecanismo de comunicación usado por MPI es la transmisión de data de un proceso a otro (pares de procesos), donde uno de ellos es el emisor y el otro es el receptor.

II.5.1.1 Modelos de comunicación

El modelo bloqueante está relacionado con el tiempo que un proceso permanece bloqueado una vez que llama a una función de comunicación, bien sea de envío o recepción de mensajes, haciendo que el proceso esté en pausa hasta tanto no finalice la operación solicitada.

El modelo no bloqueante solicita al sistema que realice una tarea e inmediatamente continua con el trabajo que venía realizando, teniendo que verificar en otro punto de la ejecución si la tarea solicitada ha culminado.

II.5.1.2 Modos de envío de mensajes

- Modo estándar: En este tipo de comunicación depende de MPI si los mensajes serán almacenados en un buffer o no. Si se utiliza el buffer, la función de envío culminará su trabajo antes que la función de recepción sea invocada. Por el contrario es posible que el buffer no esté disponible, en este caso la función de envío se bloquea hasta tanto su similar de recepción sea invocada y la data haya sido movida al receptor.
- Modo por buffer: La operación de envío puede ser iniciada y finalizar sin importar si la operación de recepción ha sido invocada. El mensaje a ser enviado es almacenado en un buffer local suministrado por la aplicación. El espacio ocupado del buffer es liberado cuando el mensaje es transferido a su destino o cuando el buffer de envío sea cancelado.
- Modo Sincrónico: la operación de envío se puede iniciar sin necesidad que la de recepción sea invocada. Sin embargo para que el envío sea satisfactorio es necesario que la operación de recibo sea iniciada y haya comenzado la recepción del mensaje enviado.
- Modo listo (ready): para que el envío pueda ser iniciado también la operación de recibo debe estar lista para la recepción del mensaje, de lo contrario la operación es errónea y el resultado es indefinido.

La comparación entre PVM y MPI puede ser consultada en el apéndice D.

CAPÍTULO III
MARCO METODOLÓGICO

Debido a las características del proyecto se utilizó para el desarrollo la metodología denominada Prototipo Evolutivo, la cual está basada en la elaboración de prototipos o versiones preliminares de un nuevo producto, donde se incorporan algunas características del sistema final, permitiendo evaluar los requisitos y probar diferentes opciones de diseño.

Este modelo consiste en definir los requerimientos iniciales, codificarlos y revisar el resultado con el usuario, lo cual permite una retroalimentación del mismo, ofreciendo la posibilidad de realizar los cambios sugeridos hasta que se garantice su satisfacción. Este tipo de versiones suelen tener funcionalidad reducida, incrementándose paulatinamente y haciendo uso de refinamientos sucesivos de las especificaciones del sistema.

Debido a las características antes citadas, se considero conveniente aplicar esta metodología para la elaboración de este proyecto puesto que cada versión o prototipo permite una mejor comprensión del problema para así ofrecer la solución más idónea. Adicionalmente el Prototipo Evolutivo permite identificar errores u omisiones en los requerimientos propuestos a través de la retroalimentación ofrecida por los usuarios del sistema. Además ofrece la posibilidad de realizar cambios fácilmente y de agregar nuevas funcionalidades en cada prototipo.

Las etapas del ciclo de vida que ofrece la metodología son las siguientes:

- Análisis de requerimientos: En esta etapa se determinan cuales son las funcionalidades que va a tener la aplicación a desarrollar, obteniendo la información a través de entrevistas, encuestas o documentos que ilustren con mayor claridad las necesidades del cliente. Entre los requerimientos a determinar se encuentran los de entorno, que se refieren a los ambientes de programación a utilizar, sistemas operativos, manejadores de bases de datos, entre otros. Los requerimientos funcionales que permiten determinar lo que el sistema debe hacer. Los de desempeño que involucran variables como velocidad, frecuencia, recursos y transacciones.

- Diseño global: Esta etapa se centra en traducir los requerimientos previamente establecidos, es decir, consiste en plasmar los requerimientos en forma de datos. Para ello se determina la

estructura de datos, arquitectura del sistema, detalles de los procedimientos y la interfaz de usuarios. Además se determinan las transacciones realizadas por la aplicación, entradas, salidas, detalles de interfaces, de seguridad, bases de datos y archivos.

- Desarrollo del Prototipo: El desarrollo consiste en la codificación o traducción de los requerimientos siguiendo los parámetros de diseño planteados anteriormente, transformándolo a un lenguaje que sea comprendido por la máquina.

- Pruebas: esta etapa está basada en procesos lógicos internos del software. Esta prueba consiste en comprobar que cada una de las sentencias del mismo se encuentren correctas, además de revisar que todas de las entradas y salidas generen respuestas que se adecuen a los requerimientos del sistema. Adicionalmente se incluyen las pruebas que involucran al usuario siendo de gran importancia debido a las mejoras que estos aportan.

- Entrega final del sistema: una vez finalizado el prototipo, cumplidos los objetivos y si se considera que se han satisfecho todos los requerimientos planteados por el usuario, se puede declarar el producto como funcional y se procede a su implantación.

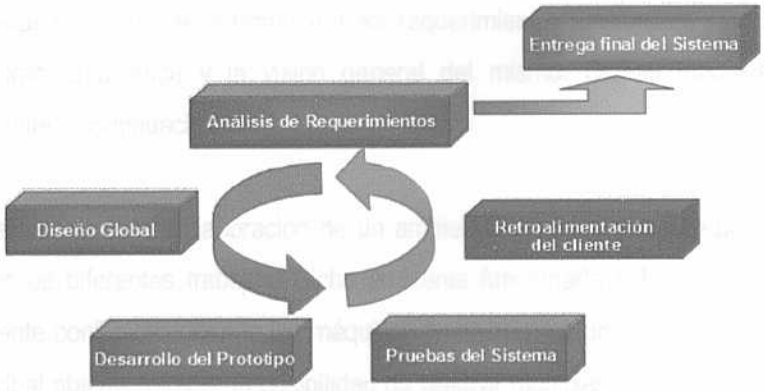

Figura 3.1 Metodología de Prototipo Evolutivo
Fuente: Elaboración propia

CAPÍTULO IV
DESARROLLO

En este capítulo se explican en detalle cada una de las actividades realizadas en el ensamblado, instalación y configuración del cluster, así como en la construcción de la aplicación de acuerdo a las etapas de la metodología antes citada.

IV.1 Desarrollo del primer prototipo

Esta etapa describe en forma detallada todos los pasos llevados a cabo para la elaboración del primer prototipo, ofreciendo una visión general del proyecto, lo cual sirvió como base para la incorporación de las mejoras aportadas por el cliente en los prototipos subsiguientes.

IV.1.1 Análisis de requerimientos

Para el análisis de requerimientos fue necesario el levantamiento de información en lo que se refiere al alcance del proyecto y sus funcionalidades. Con este fin se llevaron a cabo una serie de entrevistas con el tutor quien junto con un grupo de personas serían los usuarios finales de la plataforma, de estas reuniones de determinaron los requerimientos funcionales y no funcionales del proyecto, así como el alcance y la visión general del mismo. Dichas funcionalidades serán explicadas en detalle a continuación.

En forma general se planteó la elaboración de un ambiente de procesamiento paralelo distribuido para la ejecución de diferentes trabajos. Dicho ambiente funcionaría sobre la arquitectura de un cluster debidamente configurado donde las máquinas trabajen para un fin común. Esta plataforma tiene como principal objetivo ofrecer la posibilidad de realizar múltiples tareas de forma simultánea, cada una de ellas procesando diferentes cantidades y tipos de datos. Para ello se debía distribuir la data a procesar en varias máquinas, con el fin de aligerar la carga que implicaría su ejecución simultánea en un solo procesador.

De las reuniones realizadas se dictaminaron una serie de propósitos que debían cumplir tanto el cluster como la plataforma a desarrollar:

- El cluster debía ser del tipo Beowulf, lo cual significa que estaría conformado por un conjunto de computadores trabajando bajo el sistema operativo Linux.
- Las máquinas del cluster serían utilizadas tanto como estaciones de trabajo como para el procesamiento paralelo, lo que significa que debía ser del tipo *no dedicado,* es decir, cada computador puede también ejecutar sus propios procesos.
- Debía ofrecer la posibilidad de compartir archivos y de administrar de forma centralizada las cuentas de los usuarios.
- La plataforma debía brindar transparencia al usuario haciéndolo sentir que está trabajando en una sola máquina, abstrayéndolo de aspectos como la ubicación de la data que se está procesando, de la reubicación de datos en caso de fallos y permitirle tener acceso a la plataforma sin importar si es de forma remota o localmente en el servidor.
- Ofrecer disponibilidad, dando al usuario la seguridad de que el procesamiento será realizado con éxito puesto que la aplicación debe ofrecer mecanismos preventivos para que la ejecución continúe en caso de fallas en la red o en alguna de las máquinas del cluster.
- Sólo los usuarios autorizados podrían tener acceso a la plataforma.
- Evitar en la medida de lo posible sobrecargar de trabajo los procesadores.
- Se acordó la existencia de cuatro procesos con funciones muy específicas:
  - ✓ Proceso Inicializador: encargado de iniciar la aplicación y permanecer a la espera de mensajes provenientes de los demás procesos que se estén ejecutando.
  - ✓ Proceso Central: su función sería autenticar al usuario que desee acceder a la aplicación y verificar la disponibilidad de máquinas para la ejecución.
  - ✓ Proceso Maestro: encargado de distribuir la data entre las máquinas disponibles, monitorear la ejecución de las tareas, reasignar la data en caso de fallos y terminar la ejecución del usuario.
  - ✓ Proceso Esclavo: cuya función se basa en procesar la data que le fue asignada y devolver los resultados.

En base a estos requerimientos se procedió a la elaboración del diseño del cluster y de la plataforma, que servirían como base para la construcción del primer prototipo, el cual contaría con algunas de las funcionalidades antes citadas.

IV.1.2 Recursos de Hardware necesarios

Inicialmente se destinaron para la elaboración del cluster tres máquinas que servirían como base para un futuro proyecto de mayor alcance. Teniendo como premisa las características y funcionalidades que debe tener el cluster fue necesario contar con los siguientes recursos:

- Cable UTP (Unshielded twisted Pair), con conectores RJ45.
- Switch de 8 puertos.
- Un punto de red.

Las máquinas del cluster deben poseer:

- Tarjetas de Red (10/100Mbps), dos en el servidor y una en cada cliente.

IV.1.3 Elección del ambiente de trabajo

Durante el análisis de requerimientos se fijaron las herramientas a utilizar para la elaboración del cluster y de la plataforma. A continuación se describen cada una de ellas:

El cluster por ser del tipo Beowulf debía estar basado en el sistema operativo Linux. Debido a que la máquina que sería empleada como servidor poseía Fedora Core 4, se sugirió la instalación del mismo para las máquinas clientes pero de forma más sencilla.

Para poder acceder a ficheros remotos se decidió emplear el protocolo NFS (Network File System), el cual es un sistema de archivos distribuidos para ser utilizado en el entorno de una red. En cuanto a la administración de los usuarios de forma centralizada, se decidió emplear el sistema NIS (Network Information System), el cual genera una base de datos centralizada de usuarios o grupos de usuarios que podrán tener acceso a cualquiera de las máquinas del cluster.

En cuanto a la plataforma el tutor sugirió que se empleara como lenguaje de programación C, ya que a pesar de ser un lenguaje de bajo nivel es posible realizar implementaciones óptimas. Además

es altamente portable puesto que existen compiladores para casi todos los sistemas operativos existentes.

Se dictaminó que para la comunicación entre los procesos era necesaria la utilización de librerías de paso de mensajes, por esto, el tutor sugirió las dos librerías por excelencia utilizadas en la actualidad: PVM (Parallel Virtual Machine) y MPI (Message Passing Interface).

Para poder seleccionar la librería más adecuada para la situación planteada, se realizó una investigación teórica en base a las ventajas y funcionalidades que ofrecen tanto PVM como MPI. Las características decisivas y que rigieron la selección fueron las siguientes:

Características	PVM	MPI
Portabilidad	Media	Alta
Interoperabilidad	Alta	Media
Tolerancia al Fallos	Sí	No
Control de Procesos	Sí	MPI-2: Sí
Control Código Fuente	Dinámico	Estático
Velocidad de comunicación	Lenta	Rápida
Comunicación asíncrona	Sí	Sí
Determinar dinámicamente recursos del sistema	Sí	No
Topología de paso de mensajes	NO	Sí

Tabla 4.1 Cuadro comparativo PVM - MPI

Fuente: Elaborado por el autor

A pesar que MPI es altamente portable y ofrece alta velocidad de comunicación, la librería seleccionada fue PVM debido a dos factores muy importantes: el primero de ellos es la tolerancia a fallos, PVM detecta cuando una de las computadoras que se encuentran en la máquina virtual, es inalcanzable por diferentes razones, quedando de parte del programador elaborar su aplicación de forma que sea tolerante a fallos, cosa que en MPI no es posible.

El segundo de los factores decisivos fue el control dinámico de los recursos del sistema, esto se debe a que para la plataforma a desarrollar, resultaba de gran importancia determinar cuales máquinas del cluster se encontraban ocupadas al momento de realizar la asignación de carga de trabajo con el fin de no sobrecargar los procesadores del cluster. Este mecanismo ofrecido por la PVM no es completamente confiable, puesto que es posible que después de haber consultado los recursos del sistema, ocurra una falla o culmine alguna de las ejecuciones y se asuma que el recurso continúa ocupado, sin embargo es el único mecanismo disponible al menos en las dos herramientas estudiadas.

Como manejador de base de datos se seleccionó MySQL por ser un manejador de libre distribución y de fácil adquisición.

IV.1.4 Análisis

Una vez capturados los requerimientos funcionales y no funcionales del sistema, se procedió a analizar en detalle cada uno de ellos, con lo cual se obtuvo como resultado los diagramas de flujo de procesos. Dichos diagramas muestran de forma gráfica las tareas que debe realizar cada proceso para cumplir sus objetivos.

A continuación se muestra el diagrama de flujo de procesos para el "Proceso Inicializador" (ver Figura 4.1).

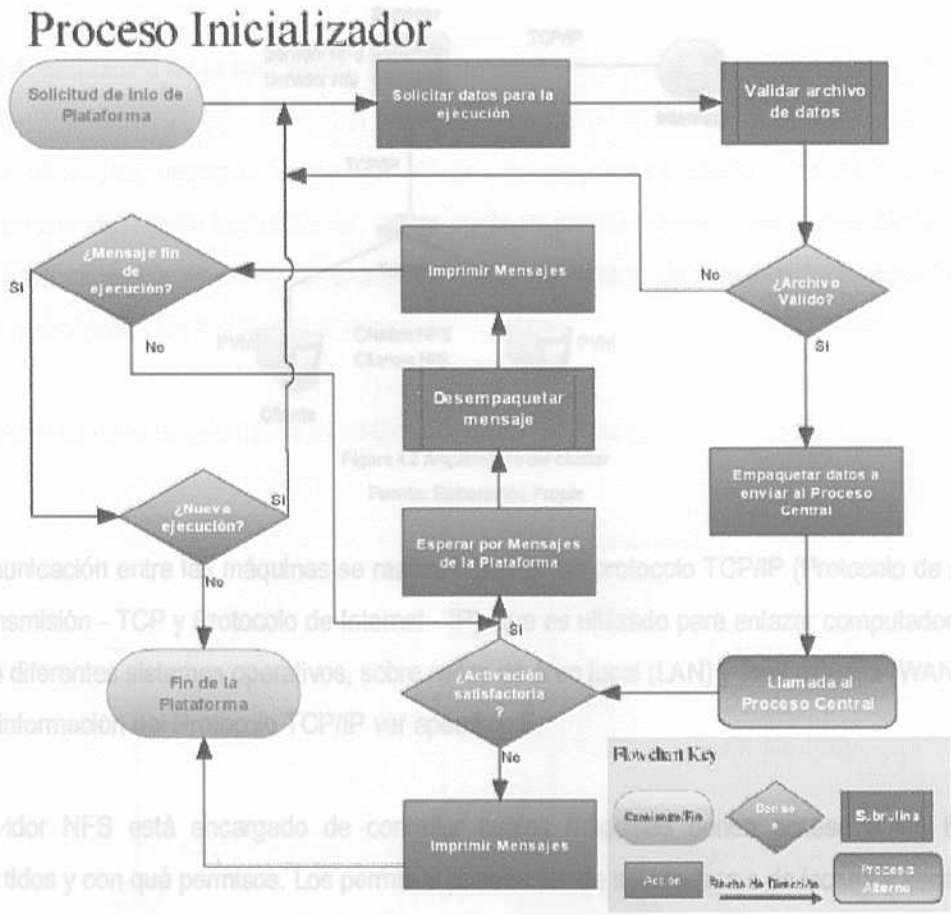

Figura 4.1 Diagrama de flujo de procesos del "Proceso Inicializador"

Fuente: Elaboración propia

El resto de los diagramas de flujo de procesos pueden ser visualizados en el apéndice E.

IV.1.4.1 Arquitectura del cluster

En base al análisis realizado se decidió emplear una arquitectura cliente – servidor debido a las configuraciones que requieren las herramientas seleccionadas para compartir ficheros de forma remota y para administración de usuarios de forma centralizada (NFS y NIS respectivamente).

La arquitectura del cluster se muestra en la siguiente figura:

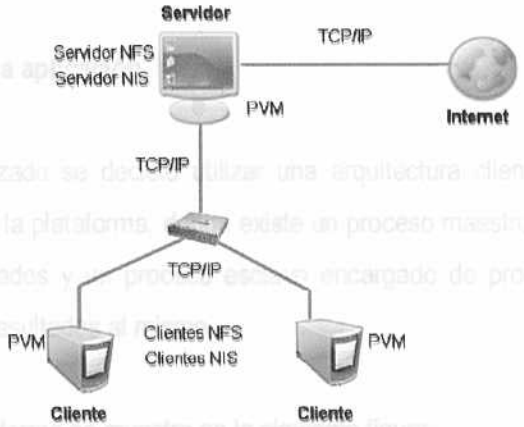

Figura 4.2 Arquitectura del cluster

Fuente: Elaboración Propia

La comunicación entre las máquinas se realiza a través del protocolo TCP/IP (Protocolo de Control de Transmisión - TCP y Protocolo de Internet - IP), que es utilizado para enlazar computadoras que poseen diferentes sistemas operativos, sobre redes de área local (LAN) y área extensa (WAN). Para mayor información del Protocolo TCP/IP ver apéndice F.

El servidor NFS está encargado de controlar cuáles máquinas tienen acceso a los ficheros compartidos y con qué permisos. Los permisos pueden ser de sólo lectura o de lectura y escritura.

El servidor NIS tiene como finalidad proveer una base de datos centralizada de usuarios del cluster, gracias a este sistema es posible incorporar y eliminar usuarios únicamente en el servidor, sin necesidad de ser modificados en cada máquina cliente.

El cliente NFS está encargado de montar el fichero remoto en la máquina cliente de tal manera que pueda ser manipulado como si fuese local pero teniendo en cuenta los permisos otorgados por el servidor.

El cliente NIS está encargado de localizar al maestro que ofrece el servicio NIS para así poder solicitar información de usuarios al servidor.

Finalmente la PVM (Parallel Virtual Machine) que tiene como finalidad montar una máquina virtual paralela que se basa en el paso de mensajes para comunicar procesos y dar la sensación que se

trabaja ante un único supercomputador. Será la principal herramienta utilizada por la plataforma para llevar a cabo sus funciones.

IV.1.4.2 Arquitectura de la aplicación

En base al análisis realizado se decidió utilizar una arquitectura cliente – servidor debido a la estructura que debe tener la plataforma, donde existe un proceso maestro encargado de distribuir la data y mostrar los resultados y un proceso esclavo encargado de procesar la data recibida del maestro y devolverle los resultados al mismo.

La arquitectura de la plataforma se muestra en la siguiente figura:

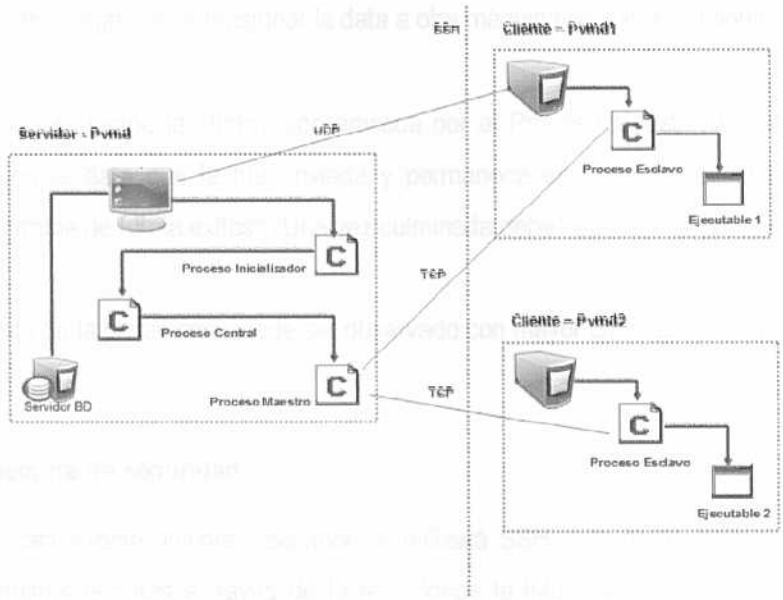

Figura 4.3 Arquitectura de la Plataforma
Fuente: Elaboración Propia

La comunicación entre los procesos se realiza a través del protocolo TCP y entre los Pvmd (demonios de la PVM) se realiza a través del protocolo UDP. Cada uno de los componentes de la arquitectura serán explicados con mayor detalle a continuación:

- Proceso Inicializador: tiene la función de interactuar con el usuario, esto lo hace solicitando los datos necesarios para la ejecución, tales como el nombre de usuario y contraseña, ruta y nombre del archivo que contiene la data a procesar y el número de máquinas que necesita para

la ejecución. Una vez recolectada la información debe validar el archivo de datos, llamar al Proceso Central y permanecer en espera de mensajes provenientes del resto de los procesos.

- Proceso Central: como paso inicial debe autenticar el usuario, una vez validado procede a verificar la disponibilidad de máquinas. De haber máquinas disponibles las asigna e inicia al Proceso Maestro.

- Proceso Maestro: debe empaquetar la data a ser enviada a las máquinas asignadas, una vez empaquetada es enviada a los procesos esclavos y solicita mensajes de culminación de tareas, lo cual debe notificar al Proceso Inicializador. Después de haber distribuido el trabajo debe permanecer en constante monitoreo de las tareas ejecutándose, a fin de detectar fallas provenientes de la red, de los esclavos o del propio servidor. De presentarse alguna de las fallas antes expuestas, debe reasignar la data a otra máquina para que continúe la ejecución.

- Proceso Esclavo: recibe la información enviada por el Proceso Maestro, ejecuta el programa solicitado con la data que le fue enviada y permanece en espera para asegurarse que la ejecución termine de forma exitosa. Una vez culminada debe notificarlo al proceso Inicializador.

El funcionamiento de la plataforma puede ser observado con mayor claridad en el apéndice G.

IV.1.4.3 Arquitectura de seguridad

Para la comunicación entre cliente - servidor se utilizará SSH, que es un protocolo que permite acceder a máquinas remotas a través de la red, donde la información que se intercambia viaja cifrada de forma asimétrica, es decir, con la utilización de claves públicas y privadas.

Para mayor información acerca del protocolo SSH ver apéndice H.

IV.1.5 Diseño global del primer prototipo

Con la información recolectada en las fases anteriores, se procedió a realizar el diseño del cluster y del sistema, en esta etapa se explica en detalle la forma como serán construidos cada uno de los componentes de las arquitecturas antes mencionadas.

Para este primer prototipo se implementarán los procesos sin mucho nivel de detalle, esto quiere decir que algunas funcionalidades serán agregadas en prototipos subsiguientes.

IV.1.5.1 Diseño de la red

Tal como se explicó anteriormente, para la comunicación de las máquinas del cluster se empleará el protocolo TCP/IP, por esto fue necesario el diseño de la red, determinando clase de direcciones IP y rango.

Para la elaboración de la red se decidió emplear las direcciones clase C, el rango a utilizar será: 192.168.1.1 – 192.168.1.254, donde la dirección 192.168.1.0 se utilizará para identificar la red y la 192.168.1.255 será reservada para el broadcast.

IV.1.5.2 Diseño de la comunicación entre los componentes de la plataforma

Como se explicó en la fase anterior, la plataforma tendrá una arquitectura cliente – servidor, dividida en cuatro procesos que se distribuyen de la siguiente manera:

- **Servidor:** Proceso Inicializador, Proceso Central, Proceso Maestro.
- **Cliente:** Proceso Esclavo.

Tal como se mencionó anteriormente la comunicación entre dichos procesos será a través del paso de mensajes, para ello la PVM emplea el protocolo TCP, el cual establece una canal de comunicación entra las tareas que es fiable puesto que se encuentra libre de errores y no se experimentan pérdidas ni duplicaciones.

A un nivel más interno la PVM provee dos mecanismos de comunicación, síncrona y asíncrona. El tipo a utilizar para el envío y recepción de mensajes entre los procesos será la asíncrona, esto se debe a que el remitente no tiene que esperar por la respuesta del destinatario, por tanto los dos extremos no necesitan trabajar de forma sincronizada, es decir, permite que el proceso que recibe o envía el mensaje no se bloquee hasta tanto esto no ocurra, sino que por el contrario puede continuar con su ejecución permaneciendo en constante monitoreo para detectar que ha arribado algún mensaje.

IV.1.5.3 Diseño del paradigma de programación paralela a utilizar

En general las aplicaciones para procesamiento paralelo que emplean librerías de paso de mensajes, se encuentran clasificadas en dos grandes grupos según el paradigma de programación paralela que empleen, estos grupos son: paralelismo funcional y paralelismo estructural o a nivel de datos.

El paralelismo funcional implica que las diferentes partes de un programa estén realizando distintas tareas de una forma concurrente y a la vez cooperativa. Mientras que el paralelismo estructural significa que se realiza una ejecución de procesos paralelos con idéntico modo de operación pero sobre distintos conjuntos de datos.

Según los paradigmas antes mencionados se decidió diseñar la plataforma de tal manera que empleara el paradigma funcional de dos formas diferentes según el nivel de la ejecución en el que se encuentre. A nivel del servidor es utilizado por los procesos Inicializador, Central y Maestro, puesto que estos se encontrarán trabajando de forma cooperativa pero sobre diferentes partes del programa para llegar a un fin común, para este caso en un nivel de la ejecución se encontrarán trabajando en forma paralela e intercambiando información el proceso inicializador con el proceso central y en una etapa posterior, el inicializador con el maestro. Por otro lado, a nivel de los clientes se emplea el mismo paradigma pero en su caso más extremo, donde el mismo programa se ejecuta en diferentes procesadores (proceso esclavo), pero procesando diferentes conjuntos de datos. La diferencia de este paradigma con el paradigma estructural es que en este último debe existir comunicación entre los esclavos, mientras que en este caso cada uno de ellos trabaja de forma independiente al otro.

El motivo por el cual se decidió diseñar la plataforma de la manera antes mencionada se debe a que este paradigma es altamente recomendado cuando el número de tareas es superior al número de procesadores disponibles o cuado no se conoce con exactitud el número de trabajos a realizar desde el comienzo de la ejecución. Además este tipo de balance de carga dinámico permite a la aplicación adaptarse fácilmente a posibles cambios del sistema en lo que se refiere a reconfiguraciones de los recursos existentes, puesto que responde de forma muy adecuada ante la posible pérdida de alguno de los esclavos, facilitando la elaboración de aplicaciones tolerantes a fallos.

IV.1.5.4 Diseño del archivo de datos

El archivo de datos debe contener toda la información necesaria para que la plataforma pueda realizar su trabajo. Este archivo es elaborado por el usuario y debe tener la siguiente estructura:

<ruta del ejecutable><nombre del ejecutable> <ruta del archivo de parámetros><nombre del archivo de parámetros>

En la siguiente figura se puede observar un ejemplo del archivo de datos:

```
MisDatos.txt

/root/MyFiles/ejecutable1 /usr/MisParametros/parametros1.txt
/mnt/Share/MisArchivos/ejecutable2 /mnt/Share/Param/parametros2.txt
/mnt/Share/MisArchivos/ejecutable2 /mnt/Share/Param/parametros3.txt
```

Figura 4.4 Estructura del archivo de datos

Fuente: Elaboración propia

Como se pudo observar en la figura 4.4 se solicita la ruta y nombre de dos archivos, el primero de ellos le indica al proceso esclavo que archivo debe ejecutar y el segundo indica la ubicación de la data con la que lo va a ejecutar. Esto quiere decir que cualquier código que se desee ejecutar en la plataforma deberá recibir como parámetro la ruta de un archivo con el cual va a realizar todos sus cómputos.

IV.1.5.5 Diseño del Proceso Inicializador

El proceso inicializador está encargado de encaminar las solicitudes de ejecución del usuario y permanecer en constante interacción con el mismo, comunicándole las acciones tomadas por la aplicación durante su ejecución. Gracias al análisis realizado en la etapa anterior acerca de las funcionalidades que debía cubrir este proceso, se pudo tener una idea más clara del diseño que éste debe tener.

Este proceso consta de varias partes o subtareas, la primera de ellas es solicitar los datos de la ejecución al usuario, entre estos se encuentra el nombre de usuario y la contraseña, ruta y nombre del archivo que contiene los datos a procesar y cantidad de máquinas necesarias para la ejecución. La interfaz con el usuario es a través del terminal, la razón por la que se hizo de esta forma tan sencilla se debe a que se pretende evitar a toda costa el exceso de carga en los CPU del cluster.

Una vez recolectada la información se procede a validar el archivo de datos, esto implica únicamente verificar que el archivo exista y que se encuentre en la ruta especificada por el usuario.

De ser válido el archivo, el proceso inicializador debe realizar su segunda tarea que se basa en construir o empaquetar el conjunto de datos recolectados en la tarea anterior, los cuales van a ser utilizados por el proceso central. Una vez empaquetados los datos pasa a su siguiente tarea que consiste en invocar a una subrutina encargada de ejecutar al proceso central enviándole los datos empaquetados.

Finalmente se pueden tomar dos acciones de acuerdo a lo ocurrido en la tarea anterior, la primera de ellas se desencadena al presentarse un error al hacer la llamada al proceso central, esto significa que ha ocurrido una falla en la máquina virtual, lo cual implica que pudo haber ocurrido un problema a nivel del pvmd (demonio de la PVM en el servidor), como se mencionó anteriormente, la PVM puede tolerar la falla de cualquiera de sus esclavos pero nunca del servidor, por esto lo más recomendable es finalizar la ejecución de la plataforma.

La segunda de las acciones a tomar se desencadena al haber iniciado exitosamente al proceso central, entonces la próxima tarea del inicializador será permanecer a la espera de mensajes provenientes del resto de los procesos de la plataforma. Para dicha labor se deben invocar rutinas de recepción y desempaquetado de mensajes que permitirán mostrarlos al usuario por pantalla.

Por último, al culminar la ejecución se ofrece la opción al usuario de continuar ejecutando con un nuevo conjunto de datos o de cerrar el ambiente.

IV.1.5.6 Diseño del Proceso Central

El proceso central tiene una labor corta puesto que es el único de los procesos que no permanece activo durante toda la ejecución. Se ejecuta paralelamente al inicializador, su función principal se basa en validar al usuario y verificar la disponibilidad de maquinas.

Para la verificación de máquinas disponibles el proceso central se debe asegurar que no estén participando en otra ejecución, esto quiere decir que sólo se permitirá una ejecución por máquina a la vez, esta decisión fue tomada basándose en que los componentes de hardware de las máquinas esclavas son antiguos y por lo tanto no se desean sobrecargar de trabajo.

Para dicha verificación se invocará una rutina que permitirá ver que hosts de la máquina virtual están realizando alguna tarea, de acuerdo a esto y al número de máquinas solicitado por el usuario se tomarán dos acciones, la primera de ellas es otorgar tantas máquinas como sean solicitadas por el usuario siempre y cuando el número solicitado no sobrepase la mitad de máquinas del cluster más una, de lo contrario se asignará el máximo permitido. De hecho la plataforma se considerará apta para iniciar un procesamiento cuando exista al menos una máquina disponible para la ejecución.

La segunda acción es desencadenada cuando no existan máquinas disponibles, es allí cuando se invoca una rutina para envío de mensaje que se encargará comunicarle al proceso inicializador y por ende al usuario que no existen máquinas disponibles para dicha ejecución y se finalizará la plataforma con el fin de no mantener ocupados los recursos, puesto que las ejecuciones que se estén llevando a cabo pueden tomar largo tiempo para su culminación.

De haber al menos una máquina disponible se invoca al proceso maestro enviándole el conjunto de máquinas con el cual cuenta para la ejecución, finalizando así la labor del proceso central. Al igual que en el proceso inicializador si la llamada al maestro no es satisfactoria se considera que hubo una falla en el servidor y por lo tanto se cierra la plataforma.

IV.1.5.7 Diseño del Proceso Maestro

El proceso maestro es el más complejo de la plataforma puesto que está encargado de distribuir la data entre las máquinas que le fueron asignadas y verificar que no ocurran fallos, de ocurrir debe reasignar la data a una nueva máquina.

Este proceso está conformado por un conjunto de tareas que debe realizar para llevar a cabo su trabajo. La primera de ellas consiste en realizar una lectura del archivo de datos a fin de desglosar la información que en él se encuentra para distribuirla entre las máquinas esclavas. Una vez desglosada la información procede a empaquetarla y llama a una rutina que tiene como finalidad colocar en una cola cada uno de los datos a ser procesados, esta última acción es realizada para tener una medida de control en caso que el número de máquinas disponibles para la ejecución sea inferior al número de tareas a ejecutar, de esta manera se ejecuta una tarea por máquina a la vez, se espera por su culminación y se procede a enviarle un nuevo conjunto de datos.

Una vez encolada la información pasa a la siguiente tarea que consiste en activar los procesos esclavos remotamente, para esto se invoca una rutina a la cual se debe indicar en que máquina se desea activar al proceso esclavo y con qué datos. Esta tarea desencadena dos acciones, la primera de ellas se origina cuando falla la activación del proceso esclavo, esto puede suceder por varias razones, por fallas en la red, por fallas en la máquina esclava o por no localizar al proceso esclavo en la máquina esclava. Si esto sucede, la data que no se pudo enviar es colocada al final de la cola, la máquina que experimentó el error es descartada para próximos envíos y se continúa con la asignación de data. Si por el contrario la activación es satisfactoria, se continúa asignando data a otras máquinas, de agotarse el grupo de máquinas disponibles, se espera a que alguna de ellas culmine su tarea actual para proceder a asignarle otra ejecución. El usuario puede hacer un rastreo de los eventos que ocurren en la plataforma gracias al proceso inicializador, puesto que éste le comunica lo que va ocurriendo en su ejecución.

Si todas las máquinas asignadas fallan antes de la ejecución, se comunica al usuario que ha ocurrido un error en las máquinas esclavas y que no se pudo iniciar el procesamiento.

Como siguiente tarea el proceso maestro solicita un mensaje de culminación a los procesos esclavos a fin de llevar un control de cuantas tareas han culminado satisfactoriamente y cuantas restan, para saber así cuando hayan culminado todas para proceder a terminar con su trabajo. Para esto el maestro realiza un monitoreo constante tanto de la tareas ejecutándose como de las máquinas que le fueron asignadas con la meta de detectar si ha ocurrido una falla en las máquinas esclavas, de ocurrir, se encola la data tal y como se explicó anteriormente.

Nuevamente si todas las máquinas que se le asignaron fallan durante la ejecución, se invoca a una rutina que permite darle la libertad a la PVM de elegir la máquina más adecuada para realizar el trabajo. La PVM cuenta con un mecanismo que ofrece la posibilidad medir según los recursos del computador, cuál es el más apto para desempeñar una labor en un momento dado, aun cuando todas las máquinas se encuentren ocupadas.

La única manera que el proceso maestro aborte una ejecución después de iniciada, es que ocurra una falla en el servidor o en todas las máquinas del cluster.

Una vez culminadas todas las ejecuciones, el proceso maestro comunica al inicializador que ha culminado exitosamente su labor.

IV.1.5.8 Diseño del Proceso Esclavo

Finalmente el proceso esclavo es el más sencillo de la plataforma, está encargado de recibir la data que le fue enviada por el maestro y realizar la ejecución.

Para llevar a cabo estas labores el proceso esclavo invoca a una función que le permite activar la ejecución del archivo que le fue asignado con el conjunto de data que le fue enviado. Esta activación puede desencadenar dos acciones, la primera es que no consiga el archivo a ejecutar, cosa que debe indicar al proceso maestro para que este tome las acciones pertinentes y termina así con su trabajo. La segunda acción es desencadenada cuando la activación es satisfactoria, para ello el proceso esclavo permanece a la espera con el fin de asegurarse que la ejecución culmine satisfactoriamente. Una vez terminada invoca a una rutina para comunicar del hecho al proceso maestro y termina con su labor.

IV.1.6 Implementación del primer prototipo

En esta etapa se pretende explicar con detalle la traducción de la información recolectada en las etapas anteriores a código fuente de forma que pueda ser comprendido por la máquina.

IV.1.6.1 Herramientas seleccionadas para la implementación

Para la elaboración del primer prototipo se emplearon diferentes herramientas, cada una de ellas encargadas de un área en específico. A continuación se especifican cada una de ellas:

Herramienta	Descripción
Fedora Core 4	Es una distribución Linux desarrollada por la comunidad Fedora y promovida por la compañía Red Hat. Tiene como objetivo proveer un sistema operativo de libre distribución.
NFS v4 (Network File System)	Es un sistema de archivos distribuido para un entorno de red de área local. Hace posible que distintos sistemas conectados a una misma red accedan a ficheros remotos como si se tratara de ficheros locales. (http://es.wikipedia.org/wiki/NFS, 2006)
ypserv 2.13 NIS (Network Information System)	Es un sistema de base de datos distribuida que reemplaza archivos de configuración duplicados en cada máquina, por un manejo centralizado de los mismos.
C	Es un lenguaje de programación sencillo pero a la vez de gran potencia y con campo de aplicación ilimitado, que ofrece economía sintáctica, control de flujo, estructuras sencillas y un buen conjunto de operadores.
GCC 4.0 (GNU Compiler Collection)	Es un software de libre distribución que ofrece un conjunto de compiladores creados por el proyecto GNU. Estos compiladores se consideran estándar para los sistemas operativos similares a UNIX. Además provee diferentes niveles de de chequeo de errores.
PVM 3.4.5 (Parallel Virtual Machine)	Es un API GNU de programación C capaz de crear una máquina virtual paralela, haciendo creer que se está ante una única supermáquina. Posee funciones de paso de mensaje para la comunicación entre procesos.

Tabla 4.2 Herramientas utilizadas para la construcción del primer prototipo

Fuente: Elaborado por el autor

IV.1.6.2 Implementación del cluster

A continuación se explican cada una de las configuraciones necesarias para la implementación del cluster:

Inicialmente se instaló en cada una de las máquinas del cluster el sistema operativo Fedora Core 4, siendo necesario seleccionar los paquetes requeridos para cada máquina en particular:

- En el servidor se instalaron componentes de escritorio personal (herramientas de oficina, herramientas de Internet, entre otros), componentes de estación de trabajo (compiladores) y de servidor (servidor NIS, NFS y manejador de Base de datos MySQL).
- En los clientes se instalaron algunas herramientas de oficina, los compiladores contenidos en la configuración de estación de trabajo y los clientes MySQL.

Posteriormente se asignaron los DNS y las direcciones IP a cada una de las máquinas, accediendo para ello a las configuraciones del sistema. La máquina servidor posee dos direcciones IP, una pública y una privada, la dirección pública es la que tiene permitido el acceso a Internet, por lo cual fue necesario activar el Firewall de manera que protegiera a la tarjeta de red con dicha dirección.

IV.1.6.2.1 Configuración del NFS (Network File System)

En este punto se explicarán paso por paso las configuraciones llevadas a cabo tanto en el servidor NFS como en cada uno de los clientes, necesarias para la puesta en funcionamientos del protocolo:

IV.1.6.2.1.1 Configuración del Servidor NFS

- Inicialmente se deben editar los archivos /etc/exports, /etc/hosts.deny, /etc/hosts.allow y /etc/hosts, los cuales deben quedar de la siguiente manera:
  - ✓ /etc/hosts contiene los DNS y direcciones IP de todas las máquinas que forman parte del cluster:

    127.0.0.1 localhost.localdomain localhost

    192.168.1.1 wales.CIDI2.ve wales

    192.168.1.2 scotland.CIDI.ve scotland

    192.168.1.3 germany.CIDI.ve germany
  - ✓ /etc/exports debe contener las direcciones IP de las máquinas con las cuales se van a compartir los archivos, así como los directorios o carpetas a compartir y su permisología. Tiene como finalidad servir como lista de control de acceso para los sistemas de ficheros que pueden ser explorados por los clientes. Dicho archivo debe quedar de la siguiente manera:

    /root/share 192.168.1.2(rw, no_root_squash)

    /root/share 192.168.1.3(rw, no_root_squash)

Donde */root/share* será la carpeta compartida, *rw* otorga a los clientes permiso de lectura – escritura y *no_root_squash* deshabilita la opción de root squashing, permitiendo conexión en modo root.

- ✓ /etc/hosts.allow contiene las máquinas que tienen acceso a las carpetas compartidas incluidas en el archivo /etc/exports. Debe quedar de la siguiente manera:

  portmap: 192.168.1.2, 192.168.1.3

  lockd: 192.168.1.2, 192.168.1.3

  mountd: 192.168.1.2, 192.168.1.3

  rquotad: 192.168.1.2, 192.168.1.3

  statd: 192.168.1.2, 192.168.1.3

- ✓ /etc/hosts.deny contiene los hosts que no poseen permiso para acceder a las carpetas compartidas, en este caso se niega el acceso a todas las máquinas que no se encuentren en el archivo /etc/hosts.allow. Quedando de la siguiente manera:

  portmap: ALL

  lockd: ALL

  mountd: ALL

  rquotad: ALL

  statd: ALL

- Iniciar el servicio NFS en las Configuraciones del sistema – Servicios, de tal manera que arranque el encender la máquina.
- Se inserta el módulo NFS en el kernel actual:

  Insmod nfs.ko

IV.1.6.2.1.2 Configuración del Cliente NFS

- Modificar el archivo /etc/hosts, el cual contiene los DNS y direcciones IP de todas las máquinas que forman parte del cluster:

  127.0.0.1 localhost.localdomain localhost

  192.168.1.1 wales.CIDI2.ve wales

  192.168.1.2 scotland.CIDI.ve scotland

  192.168.1.3 germany.CIDI.ve germany

- Activar el protocolo NFS en las configuraciones del sistema a fin de que inicie al encender la máquina.

- Se inserta el módulo NFS en le kernel actual:

  Insmod nfs.ko

- Para montar la carpeta compartida que se encuentra en el servidor desde el cliente, se debe ejecutar el siguiente comando:

  mount 192.168.1.1:/root/share /mnt

  donde la dirección IP pertenece al servidor, la carpeta /root/share es la que se encuentra compartida desde el servidor y /mnt es la carpeta destino.

- A fin de no tener que ejecutar el comando anterior cada vez que se reinicie la máquina se debe agregar la siguiente línea al final del archivo /etc/fstab:

  192.168.1.1:/root/share /mnt nfs rw 0 0

IV.1.6.2.2 Configuración del NIS (Network Information System)

En este punto se explicarán paso por paso las configuraciones llevadas a cabo tanto en el servidor NIS como en cada uno de los clientes, necesarias para la puesta en funcionamientos del protocolo:

IV.1.6.2.2.1 Configuración del Servidor NIS

- Para la configuración del NIS se instaló en el servidor el paquete ypserv 2.9, para ello se ejecutó lo siguiente:

  ./configure (en el directorio donde se encuentra el paquete)

  make

  make install

- En el archivo /etc/yp.conf incluir:

  domain ServerGB server 192.168.1.1

  ypserv 192.168.1.1

  Con ello se indica al NIS cual será la máquina que prestará el servicio, así como el nombre de su dominio (ServerGB).

- En el archivo /etc/sysconfig/network agregar:

  NISDOMAIN=ServerGB

  Con lo cual se indica a las configuraciones de la red el nombre del dominio NIS.

- Ejecutar el siguiente comando:

  Nisdomainname ServerGB

- Para iniciar el servidor NIS se ejecuta el siguiente comando:

  ypserv

- Para generar la base de datos NIS se ejecuta el comando:

  /usr/lib/yp/ypinit –m

Con esto se genera un directorio /var/yp/ServerGB que contiene la base de datos de claves y usuarios del cluster.

- Para escuchar por el servidor NIS previamente definido, se ejecuta el siguiente commando:

  /etc/rc.d/init.d/ypbind start

IV.1.6.2.2.2 Configuración del Cliente NIS

- En el archivo /etc/yp.conf incluir:

  domain ServerGB server 192.168.1.1

  ypserv 192.168.1.1

- Con ello se indica al NIS cual será la máquina que prestará el servicio, así como el nombre de su dominio (ServerGB).
- En el archivo /etc/sysconfig/network agregar:

  NISDOMAIN=ServerGB

  Con lo cual se indica a las configuraciones de la red el nombre del dominio NIS.
- Ejecutar el siguiente comando:

  Nisdomainname ServerGB

- Para escuchar por el servidor NIS previamente definido, se ejecuta el siguiente commando:

  /etc/rc.d/init.d/ypbind start

IV.1.6.2.3 Instalación y configuración de la PVM (Parallel Virtual Machine)

- Para la instalación se ejecutó el comando:

  yum install pvm_i386

  Donde yum es una herramienta que realiza una búsqueda en Internet del paquete solicitado y lo instala automáticamente.
- Se deben agregar las siguientes líneas al archivo /root/.bash_profile:

  PVM_RSH=/usr/bin/ssh

  export PVM_RSH

  PVM_ROOT=/usr/share/pvm3

export PVM_ROOT

PVM_DPATH=$PVM_ROOT/lib/pvmd

export PVM_DPATH

PVM_ARCH=LINUX

export PVM_ARCH

Esto con la finalidad que las variables de ambiente de la PVM sean seteadas automáticamente al iniciar el sistema. La PVM por defecto utiliza RSH para acceder a las máquinas remotas, sin embargo como se pudo observar en el archivo /root/.bash_profile, se seteó la variable de ambiente PVM_RSH apuntando a la ruta de SSH (*/usr/bin/ssh*), esto se debe a que este último ofrece mayor seguridad y utiliza varios algoritmos de cifrado (DES, IDEA, triple DES) y algoritmos como RSA para el intercambio de llaves.

- Se generaron claves públicas y privadas para cada host con los siguientes comandos:

  ssh wales.CIDI2.ve

  ssh-keygen –t dsa

- Se creó un archivo de texto en /root/.SSH llamado authorized_keys donde se deben colocar las claves públicas generadas en todos los hosts, esto para indicar cuales son las máquinas autorizadas.
- Cambiar en el archivo /usr/share/pvm3/conf/LINUX.def la ruta de rsh por la de ssh en al ARCHCFLAGS (/usr/bin/ssh), esto con el fin de indicar a la PVM que debe usar para la comunicación remota SSH.
- Para montar la PVM desde el nodo maestro se ejecuta el comando:

  pvm

  add scotland.CIDI.ve

  add germany.CIDI.ve

  Con el comando *add* se agregan todos los nodos clientes que formaran parte de la piscina de hosts de la máquina virtual.

Se debieron seguir los pasos anteriores en todas las máquinas del cluster.

IV.1.6.3 Implementación de la Plataforma

Para la implementación de los procesos que conforman la plataforma fue necesario hacer uso de varias librerías predefinidas. Estas librerías son las siguientes:

- **stdio.h:** Contiene los prototipos de las funciones, estructuras y tipos para manipular datos de entrada y salida.
- **string.h:** Contiene los prototipos de las funciones y estructuras de clasificación de caracteres.
- **stdlib.h:** Contiene los prototipos de las funciones, estructuras y tipos para utilidades de uso general (llamadas al sistema).
- **unistd.h:** Contiene definiciones de las funciones, estructuras y tipos para el control de procesos (llamadas al sistema).
- **fcntl.h:** Contiene funciones, estructuras y tipos que permiten el manejo de ficheros.
- **pvm3.h:** Contiene las funciones, estructuras y tipos necesarios para la utilización de la PVM.

IV.1.6.3.1 Implementación del Proceso Inicializador

Como se describió en la etapa anterior el proceso inicializador está encargado de dar comienzo a la plataforma, para ello deben realizar una serie de pasos previos que serán descritos a continuación:

Inicialmente se debe indicar a la PVM que un proceso va a entrar a la máquina virtual, para ello se invoca a la función *pvm_mytid*, que alista al proceso para su entrada otorgándole un identificar único dentro de la misma.

Una vez alistado se realiza una llamada al sistema de la siguiente forma: *gethostname(localH, 64)*, con ello se obtiene el nombre de la máquina donde se está ejecutando la plataforma, esto es utilizado por el inicializador para indicar al proceso central en qué maquina del cluster se debe iniciar.

Como próxima acción se solicitan los datos necesarios para la ejecución al usuario, una vez recolectados se llama a la subrutina *validFile*, que está encargada de validar que el archivo tenga el formato adecuado (archivo plano de texto) y que se encuentre en la ubicación suministrada por el usuario.

Una vez que se comprueba la validez del archivo se procede a invocar al Proceso Central a través de una llamada a la rutina *enviarData*, dicha rutina realiza las siguientes acciones: como primer paso debe empaquetar los datos a enviar al Proceso Central, luego se realiza una llamada a la función *pvm_spawn* la cual está encargada de crear un proceso hijo que se estará ejecutando de forma paralela al padre, para mayor información de las funciones de la PVM consultar apéndice C.

Una vez levantado el proceso hijo, el padre permanece a la espera de mensajes, para ello se emplea la función *pvm_nrecv*, esta función permite una comunicación asíncrona puesto que el receptor (en este caso el proceso inicializador), no debe bloquearse hasta que arribe el mensaje esperado, sino que por el contrario puede continuar realizando otros trabajos. Esta llamada es realizada dentro de un bucle que permanece en constante chequeo del arribo de mensajes, de ser recibido algún mensaje se invoca a la función *pvm_bufinfo*, que devuelve la información del buffer que posee el mensaje y finalmente se procede a desempaquetarlo haciendo una llamada a la función *pvm_upkstr*, la cual desempaqueta un mensaje de tipo cadena.

El proceso inicializador puede recibir dos tipos de mensajes, de información de las acciones realizadas por la plataforma o de culminación de ejecución. Si el mensaje es de culminación, se da la libertad al usuario de volver a realizar otra ejecución o de cerrar la plataforma. Cuando un proceso deba abandonar la máquina virtual se invoca a la función *pvm_exit*, que le indica a la PVM que debe eliminarlo de sus registros puesto que ha culminado su ejecución.

IV.1.6.3.2 Implementación del Proceso Central

Como primer paso el proceso central debe alistarse para la PVM invocando a la rutina *pvm_mytid*, de forma que le otorgue el tid que lo identificará en la máquina virtual tal como se explicó en el proceso anterior.

Para llevar el control de las máquinas ocupadas y disponibles se creó una estructura que almacena los nombres de los hosts junto con el identificador del demonio de la PVM corriendo en dicha máquina.

Una vez que se obtiene el identificador, se llama a la función *pvm_parent* que devuelve el tid del proceso padre (proceso inicializador), esto se hace con la finalidad de determinar quién será el encargado de recibir los mensajes a ser enviados.

Para poder identificar las máquinas que están ocupadas es necesario conocer cuales tareas se están ejecutando en la máquina virtual, para ello se utiliza la función *pvm_tasks,* la cual devuelve un apuntador a un arreglo de estructuras. Entre los valores más relevantes devueltos por la función se

encuentran: el identificador de la tarea, el identificador del padre o proceso que inició la tarea y el identificador del demonio corriendo la tarea, entre otros.

Luego se empleó la función *pvm_config,* que está encargada de devolver un apuntador a un arreglo de estructuras que almacena toda la información de la configuración actual de la máquina virtual,

Con el fin de realizar la verificación de las máquinas, se elaboró una rutina encargada de comparar los resultados obtenidos en las dos estructuras antes citadas, con el objetivo de identificar cuales máquinas están corriendo determinada tarea para proceder a clasificarla como ocupada y descartarla así de la ejecución actual.

Posteriormente, se informa al proceso inicializador de lo ocurrido, para ello se envía un mensaje al mismo con el uso de la función enviarMensaje:

```
void enviarMensaje(char msje[100],int ptid)
{
        pvm_initsend(PvmDataDefault);
        pvm_pkstr(msje);
        pvm_send(ptid, 1);
}
```

La primera función invocada (*pvm_initsend*) tiene como objetivo limpiar el buffer de envío y determinar el tipo de codificación a utilizar en la transmisión del mensaje, luego se procede a empaquetar el mensaje con la función *pvm_pkstr* y finalmente se hace el envío invocando a *pvm_send*, indicándole a qué tarea debe enviar el mensaje.

Una vez que se verificó cuales máquinas se podían utilizar, se procedió a empaquetar los nombres de las mismas de acuerdo al número solicitado por el usuario con el fin de comunicar al proceso maestro que máquinas le fueron asignadas para la ejecución.

Finalmente se hace la llamada al proceso maestro, enviándole como datos las máquinas asignadas y la ruta y nombre del archivo desde el cual se va a extraer la información para realizar el procesamiento y culmina con su trabajo invocando la función *pvm_exit*.

El proceso Central es el único proceso que no se encuentra activo hasta el final de la ejecución puesto que su labor culmina una vez asignadas las máquinas.

IV.1.6.3.3 Implementación del Proceso Maestro

Este proceso tiene la función de distribuir la data en las máquinas disponibles para su ejecución, monitorear que la ejecución se realice con éxito y reasignar la data en caso de fallas en la red o en las máquinas esclavas.

Para llevar a cabo su cometido se crearon las siguientes estructuras para la manipulación de los datos:

- Estructura que almacena las máquinas a las que se les va a enviar la data (*hosts*).
- Estructura para el almacenamiento de cada máquina con los datos que le fueron asignados (*enUso*), de esta manera se controla la asignación de la data para tomar medidas en caso de presentarse alguna falla en el ambiente.
- Estructura para encolar la data que aún no ha sido asignada, como se explicó en la etapa anterior, una máquina no deberá ejecutar simultáneamente más de una tarea puesto que sus componentes de hardware son antiguos, esto con la finalidad de no sobrecargar de trabajo los procesadores.

Al igual que en los casos anteriores fue necesario invocar la función que alista el proceso para la PVM (*pvm_mytid*). Luego se invocó una rutina encargada de la creación de la cola (*crear*) que será usada para el almacenamiento de la data.

Las máquinas asignadas fueron recibidas por el proceso maestro por parámetros, para poder manipularlas con mayor facilidad se debieron almacenar en una estructura.

Como siguiente paso se realizó la lectura del archivo de datos, los cuales fueron almacenados en la cola con el uso de la función *ponerEnCola*, cada uno de los datos será desencolado al momento en que vaya a ser enviado al proceso esclavo con una llamada a la función *frente*, la cual obtiene el primer valor almacenado en la cola y se invoca a la función *quitarDeCola* cuya finalidad es eliminar

el valor extraído de la cola y liberar memoria. Todo esto se realizará de forma cíclica hasta que la cola se encuentre vacía.

Una vez desencolado el dato es invocada la función *pvm_spawn* a la cual se le indica que active al proceso esclavo en la máquina remota asignada y con los datos desencolados. Al igual que los datos, las máquinas también son tratadas como si se encontrasen en una cola puesto que serán eliminadas de la estructura mientras estén en uso y vueltas a agregar una vez finalizado su trabajo, esto con el fin de no asignar más de un trabajo a la vez por máquina.

Después de enviar la data en todas las máquinas asignadas se inicia un monitoreo de todas la tareas corriendo, para ello se invoca la función *pvm_notify,* con ella se solicita a la PVM que indique a través del paso de mensajes cuando el proceso esclavo haya culminado.

Para la recuperación de fallas fue necesario invocar a la función *pvm_mstat,* dicha función tiene la finalidad de devolver el estatus de una máquina en la PVM. Los diferentes estatus evaluados por la misma son los siguientes: *PvmSysErr* (error del sistema en la máquina remota), *PvmNotImpl* (archivo a ejecutar con problemas en la implementación), *PvmHostFail* (fallas en el host esclavo o en la red) y *PvmNoHost* (cuando el host no se encuentra en la PVM).
Si alguno de los casos antes evaluados ocurre, la máquina que experimentó el problema es descartada para seguir ejecutando, por lo tanto no vuelve a ser agregada en la estructura que almacena las máquinas (*hosts*), mientras que esto ocurre se ubica que data fue enviada a dicho host consultado en la estructura *enUso,* de esta manera los datos son vueltos a encolar, a la espera que alguna de las máquinas culmine su ejecución actual para ser reasignado y cualquier registro del computador con problemas es descartado. Vale destacar que el monitoreo de errores es realizado desde el inicio del primer esclavo hasta la culminación del último de ellos.

Si se experimentan fallas en todos los hosts asignado se recurre a un mecanismo de emergencia mediante el cual la data será reasignada a la máquina más adecuada o será ejecutada localmente en el servidor, para ello se hace una llamada a la función *pvm_spawn,* la cual activa al proceso esclavo en la máquina que se le indique.

El proceso maestro habrá terminado una vez que reciba el acuse de culminación de todas las tareas que tenía bajo su mando. Cuando esto ocurre procede a liberar memoria invocando a la función

destruir, que tiene como objetivo eliminar la cola liberando el espacio que la misma usaba, luego notifica al proceso inicializador mediante la función *enviar_mensaje* que ha finalizado con su trabajo de forma satisfactoria y abandona la PVM empleando *pvm_exit.*

IV.1.6.3.4 Implementación del Proceso Esclavo

Este proceso es el más sencillo de la plataforma puesto que su única función es la de recibir la data, ejecutar la tarea y devolver los resultados.

Al igual que en los casos anteriores es alistado para la PVM con el uso de la función *pvm_mytid.* Hecho esto, informa al proceso inicializador a través de la función *enviarMensaje* que ha iniciado el proceso esclavo, procede a empaquetar la data a ser ejecutada y ejecuta la tarea invocando la función *pvm_spawn.*

Luego el proceso esclavo permanece a la espera de la culminación de la ejecución, para ello recurre a la función *pvm_pstat,* que tiene como objetivo monitorear la tarea que se está ejecutando, dicha verificación es realizada cada dos segundos con el uso de la función *sleep.*

Finalmente se envía la notificación al proceso inicializador a fin de que comunique al usuario que ha culminado la ejecución de dicho esclavo. Por su parte el proceso maestro se vale de la función *pvm_notify* (tal como se explicó anteriormente) para enterarse cuando una de las tareas ha terminado. Como último paso abandona la máquina virtual invocando a la función *pvm_exit.*

IV.1.7 Pruebas

IV.1.7.1 Pruebas del Cluster

En esta etapa se probaron cada uno de los protocolos, paquetes y servicios configurados a fin de asegurar su buen funcionamiento.

Para las pruebas del protocolo NFS se ejecutó el comando *mount 192.168.1.1:/root/share /mnt,* con esto fue posible montar en cada uno de los clientes el fichero compartido, permitiendo leer, borrar y escribir archivos en dicha carpeta.

El sistema NIS fue probado con el siguiente comando: */usr/sbin/yptest* desde los cliente, el cual devuelve como resultado los datos de la máquina maestro, que es la que ofrece el servicio NIS.

Finalmente para la prueba de la PVM se utilizó uno de los ejemplos de la misma llamado hello.c y hello_other.c ubicados en la carpeta /usr/share/pvm3/examples. Para ello se llevaron a cabo los siguientes pasos:

- Se compilaron ambos archivos con el comando:

  gcc –L/usr/share/pvm3/lib/LINUX hello.c –lpvm3 –o hello

  gcc –L/usr/share/pvm3/lib/LINUX hello_other.c –lpvm3 –o hello_other

- Los ejecutables generados se copiaron en la carpeta /usr/share/pvm3/bin/LINUX (en todas las máquinas).
- En la cónsola donde está activa la PVM se ejecutó el comando:

  Spawn -> hello

Se debe obtener respuesta de todas las máquinas del cluster que posean la PVM. Tal como se muestra en la siguiente figura:

```
pvm> spawn -> hello
1 t40010
from t80006c: hello world from scotland.cidi.ve
from t60006c: hello world from germay.cidi.ve
```

Figura 4.5 Salidas de prueba de la PVM

Fuente: Elaboración propia

IV.1.7.2 Pruebas de la Plataforma

Para este primer prototipo se realizaron pruebas muy sencillas a fin visualizar las respuestas obtenidas por la plataforma al presentarse diferentes situaciones.

Dichas pruebas fueron realizadas basadas en el criterio de caja negra, puesto que se enfocan en los resultados obtenidos en la ejecución de cada uno de los procesos sin tomar en cuenta el código que se encuentra por detrás de los mismos. En este caso se observarán los mensajes enviados por cada uno de los procesos al inicializador que es el encargado de mostrarlos por pantalla.

IV.1.7.2.1 Pruebas de funcionamiento

Para verificar las funcionalidades de la plataforma se elaboraron dos programas muy sencillos en lenguaje C que servirían para simular las ejecuciones que va a realizar el usuario. Dichos programas contienen un código muy simple cuyas ejecuciones demoran entre 5 y 10 segundos.

El archivo de datos que se suministró a la plataforma indicándole los ejecutables y los datos necesarios para el procesamiento se muestra en la siguiente figura:

```
/usr/share/pvm3/bin/LINUX/probando /root/param1
/usr/share/pvm3/bin/LINUX/probando /root/param2
/root/probando1 /root/param3
/root/probando /root/param3
```

Figura 4.6 Archivo de datos "test.txt"

Fuente: Elaboración propia

En el archivo test.txt mostrado en la figura 4.6 se le indica a la plataforma que debe ejecutar el archivo /usr/share/pvm3/bin/LINUX/probando con dos grupos de parámetros diferentes ubicados en los archivos /root/param1 y /root/param2. Adicionalmente se le indicó un segundo archivo /root/probando1 con un tercer grupo de parámetros para su ejecución. El último archivo suministrado (/root/probando) no se encuentra en la ubicación indicada, esto con el fin de probar la respuesta de la plataforma ante un archivo no encontrado.

Con esta prueba se buscaba verificar que la data fuese asignada y distribuida sin problemas, que la comunicación entre los procesos se llevara a cabo adecuadamente en lo que se refiere a notificaciones de tareas culminadas, errores ocurridos (archivo no encontrado) y mensajes de fin de la ejecución.

En la siguiente figura se muestran los mensajes enviados por el proceso Inicializador Servidor al proceso Inicio Plataforma al recibir la solicitud de un cliente y al validar el archivo de datos.

```
[root@wales LINUX]# ./InicioPlataforma

t40003: INICIALIZADOR SERVIDOR: procesando solicitud
t40003: Archivo válido.
```

Figura 4.7 Mensajes de Inicializador Servidor

Fuente: Elaboración propia

El resto de las salidas de la plataforma se pueden visualizar en el apéndice I.

IV.1.7.2.2 Pruebas de tolerancia a fallos

Para esta prueba se utilizó la plataforma de la manera que se explicó en la prueba anterior, con los mismos ejemplos pero con la diferencia que en este caso se provocó la pérdida de una de las máquinas esclavas durante la ejecución.

La pérdida de una máquina esclava se puede producir por fallas del sistema operativo o fallas en la red. Para ambos casos la PVM elimina al host que experimentó la falla de la máquina virtual, quedando en manos del proceso maestro detectar al host inalcanzable y tomar las medidas necesarias. En la figura 4.8 se puede observar el mensaje mostrados por la plataforma al perder a alguno de sus clientes.

```
t40011: Ocurrió un error en: scotland.CIDI.ve
```

Figura 4.8 Mensajes de Proceso Maestro
Fuente: Elaboración propia

Con esta prueba se pudo verificar la reacción de la plataforma ante la pérdida de un esclavo por diversas razones, la reasignación de la data y la reacción de la plataforma ante la pérdida de todos los hosts.

IV.2 Desarrollo del segundo prototipo

Para esta etapa ya se ha implementado el primer prototipo de la plataforma por lo tanto se puede tener una idea más clara de lo que se desea a futuro. En este caso se seguirán las mismas etapas del prototipo anterior, manteniendo las características de la aplicación desarrollada, agregando nuevas funcionalidades y reestructurando algunos detalles con la finalidad de que este sea el prototipo definitivo.

IV.2.1 Retroalimentación del cliente

- Inicialmente se concluyó que con el primer prototipo no se abstraía lo suficiente al usuario por tanto era necesario una reestructuración en la arquitectura de la plataforma.
- Adicionalmente se considera que el usuario no debe estar verificando constantemente si la ejecución culminó para iniciar una nueva, puesto que las mismas suelen ser muy prolongadas.

A continuación se explica en detalle las mejoras y nuevas incorporaciones realizadas al primer prototipo de acuerdo a la retroalimentación recibida por parte del tutor:

IV.2.2 Análisis de requerimientos

Aún cuando se mantienen las funcionalidades de cada proceso se decidió incorporar algunas modificaciones a la arquitectura de la plataforma, que a pesar de seguir siendo cliente – servidor, debe sufrir algunas reestructuraciones a nivel del proceso inicializador, la forma como el usuario accede a la plataforma y le suministra la información.

Entre los requerimientos funcionales a incorporar se encuentran:

- Se deben incluir las validaciones de usuarios autorizados y por tanto la incorporación del manejador de base de datos.
- La incorporación de un archivo de logs donde se lleve una bitácora de las acciones realizadas por la plataforma.

Entre los requerimientos no funcionales se encuentran:

- La elaboración de una librería que contenga todas las funciones y procedimientos usados por la plataforma.
- Se debe reestructurar el proceso inicializador, quedando dividido en tres procesos. Un proceso cliente que haga la solicitud del servicio y uno servidor que preste el servicio, esto con la finalidad de abstraer más al usuario, conservando el proceso inicializador que únicamente se encarga de mostrar los mensajes por pantalla.

IV.2.3 Análisis

A continuación se muestra el diagrama de flujo de procesos de uno de los nuevos procesos incorporados a la plataforma:

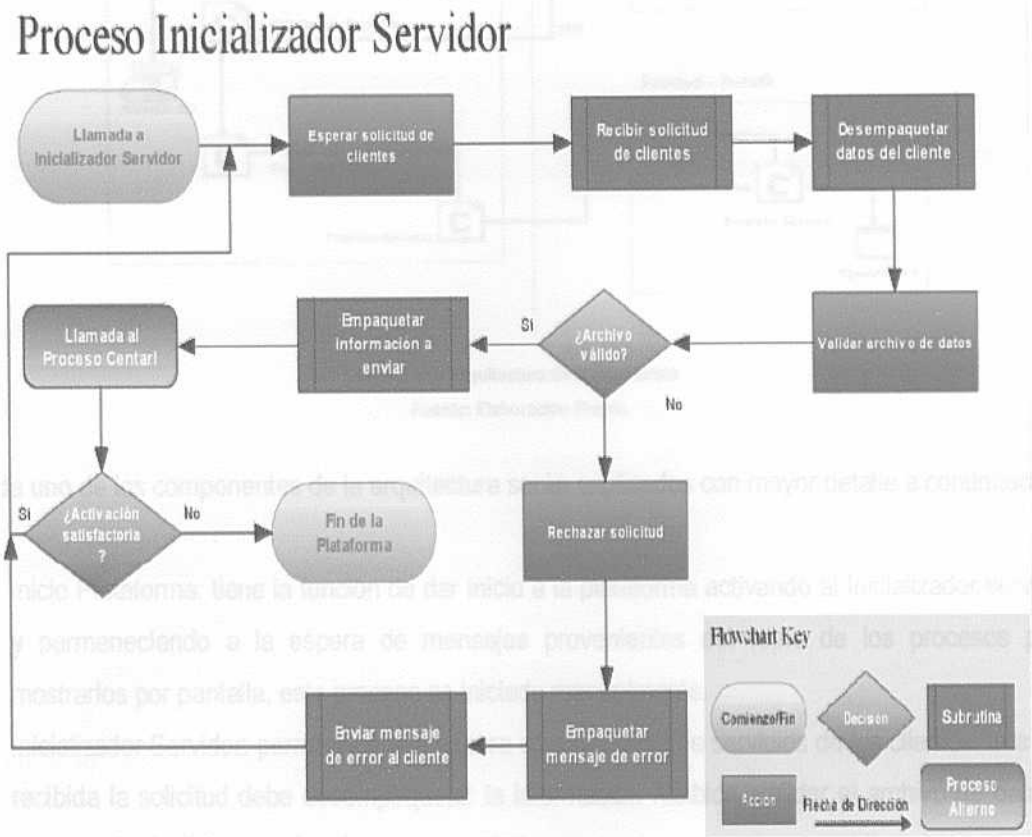

Figura 4.9 Diagrama de flujo de procesos del "Inicializador Servidor"

Fuente: Elaboración propia

El resto de los diagramas de flujos de procesos agregados a la plataforma y diagramas de secuencia pueden ser consultados en los apéndices J y L respectivamente.

IV.2.3.1 Arquitectura de la aplicación

En base a la retroalimentación recibida por parte del tutor y del análisis realizado, se decidió reestructurar la arquitectura de la plataforma, la cual puede ser observada en la figura 4.10.

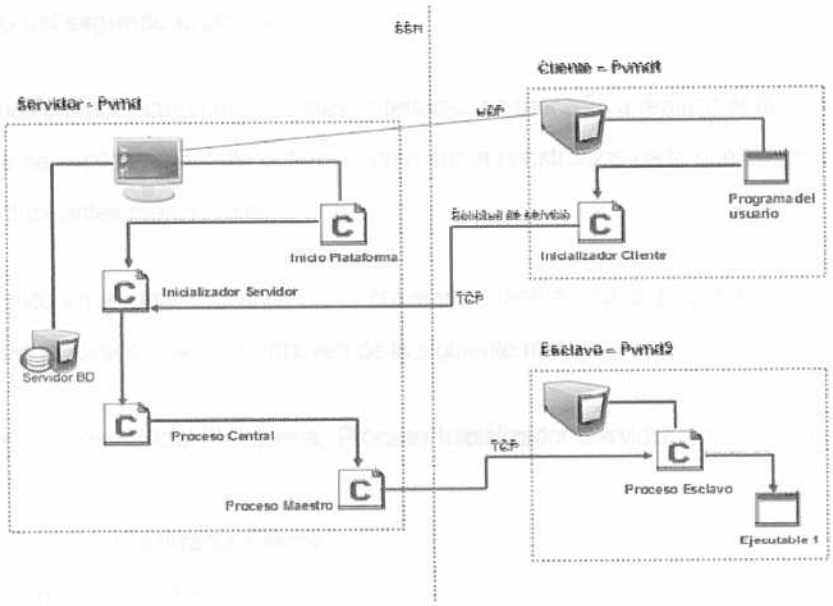

Figura 4.10 Arquitectura de la Plataforma

Fuente: Elaboración Propia

Cada uno de los componentes de la arquitectura serán explicados con mayor detalle a continuación:

- Inicio Plataforma: tiene la función de dar inicio a la plataforma activando al Inicializador servidor y permaneciendo a la espera de mensajes provenientes del resto de los procesos para mostrarlos por pantalla, este proceso es iniciado manualmente.
- Inicializador Servidor: permanece a la espera de solicitudes de servicios de los clientes, una vez recibida la solicitud debe desempaquetar la información recibida, validar el archivo de datos y empaquetar la data a enviar al proceso central.
- Inicializador Cliente: es activado por el usuario desde cualquiera de las máquinas del cluster, al iniciarlo se deben suministrar los datos para la ejecución (nombre de usuario, contraseña, archivo de datos y número de máquinas necesarias para la ejecución). Una vez activo el cliente debe enviar esta solicitud al Inicializador Servidor quien será el encargado de dar inicio al proceso de verificación y de ejecución de la data. El cliente será activado desde el código fuente del programa del usuario en el momento que necesite procesar la data.

- El resto de los procesos de la plataforma conservan las mismas funcionalidades que se explicaron en el primer prototipo.

IV.2.4 Diseño del segundo prototipo

Con la información recolectada en las fases anteriores, se procedió a realizar el diseño del sistema, en esta etapa se explica en detalle la forma como serán construidos cada uno de los componentes de la arquitectura antes mencionada.

Como se explicó en el prototipo anterior, la plataforma tendrá una arquitectura cliente – servidor, dividida en seis procesos que se distribuyen de la siguiente manera:

- **Servidor:** Proceso Inicio Plataforma, Proceso Inicializador Servidor, Proceso Central Proceso Maestro.
- **Cliente:** Proceso Inicializador Cliente.
- **Esclavo:** Proceso Esclavo.

Tal como se explicó en el prototipo anterior la comunicación entre dichos procesos será a través del paso de mensajes, para ello la PVM emplea el protocolo TCP, el cual establece una canal de comunicación entra las tareas que es fiable puesto que se encuentra libre de errores, no se experimentan pérdidas ni duplicaciones. De igual forma se utilizará el tipo de comunicación asíncrona entre los procesos que conforman la plataforma, de manera tal que el remitente no tenga que bloquearse hasta recibir respuesta del destinatario.

Además se continuará empleando el paradigma funcional tal y como se explicó en el prototipo anterior, con la incorporación de los nuevos procesos.

IV.2.4.1 Diseño del conceptual de la base de datos

En esta etapa del diseño se describe el contenido de la base de datos, esto con la finalidad de identificar entidades, atributos y relaciones, lo que va a servir para dar significado a los datos que se manejen.

A continuación se describen brevemente las tablas a utilizar:

- **User:** tiene como finalidad almacenar los datos de los usuarios que tienen acceso a la plataforma (nombre de usuario y contraseña) para así realizar la autenticación al momento de solicitar el servicio.

- **Server:** almacena los datos de Inicio Plataforma (ptid, nombre del host), el ptid es un valor entero que otorga la PVM a cada una de las tareas que los identifica en la máquina virtual, en este caso se almacena únicamente los datos del Inicio Plataforma, puesto que es el encargado de recibir los mensajes de todos los procesos, de esta manera cada uno de ellos puede conocer el identificador del proceso al que le va a enviar el mensaje.
- **Client:** Almacena los datos de los clientes (ctid, numero de máquinas para la ejecución, archivo de datos) al igual que en el caso anterior el ctid es el identificador único otorgado por la PVM para este cliente. Adicionalmente esta tabla sirve de nexo entre las dos tablas antes mencionadas, permitiendo identificar que clientes pertenecen a que usuario.

El modelo entidad – relación de la aplicación puede ser consultado en el apéndice K.

IV.2.4.2 Diseño de librerías

La plataforma contará con dos librerías, una que contiene todas las funciones de apoyo para las ejecuciones de los trabajos de los diferentes procesos que la conforman y otra librería que contendrán las funciones que realizan interacciones con la base de datos.

La razón por la que se decidió elaborar dos librerías se debe a que la máquina servidor será la única encargada de establecer conexiones con la base de datos, por tanto no es necesario que tanto el inicializador cliente como el proceso esclavo (que se encuentran en máquinas remotas) incorporen funciones a las que no podrán tener acceso, de esta forma se resguardan los datos de conexión como login, password, nombre de la base de datos, entre otros.

IV.2.4.3 Diseño del Inicio Plataforma

Este proceso será iniciado manualmente y permanecerá activo indefinidamente de forma que la plataforma esté disponible para ejecutar en el momento que el usuario lo necesite.

Su principal función es dar inicio a la plataforma activando al inicializador servidor y permaneciendo a la espera de los mensajes provenientes del resto de los procesos. Dichos mensajes serán mostrados por pantalla y almacenados en un archivo de logs que guardará un record de todas las acciones llevadas a cabo por la aplicación.

IV.2.4.4 Diseño de Inicializador Servidor

Este proceso deberá igualmente estar activo indefinidamente a la espera de solicitudes de ejecución provenientes de los clientes. Una vez recibida una solicitud, procede a desempaquetar los datos recibidos, verifica que el archivo de datos sea del formato válido (archivo plano de texto) y que se encuentre en la ubicación suministrada por el cliente. Si el archivo es inválido deberá comunicar al cliente que su solicitud ha sido rechazada. De lo contrario deberá empaquetar la data para enviarla al proceso central y dar origen a las verificaciones y procesamientos.

Si ocurre una falla al intentar dar origen al proceso central puede que haya ocurrido un error a nivel de la máquina virtual, por lo tanto la plataforma finalizará su ejecución.

IV.2.4.5 Diseño de Inicializador Cliente

Antes de describir las funcionalidades del Inicializador Cliente es necesario explicar la forma que el usuario montará dicho cliente.

Tal y como se mencionó en la fase de análisis de requerimientos, el usuario no deberá estar a la espera que culmine su ejecución para iniciar otra, es por esto que se decidió que cualquier usuario que desee acceder a la plataforma deberá realizarlo haciendo una llamada a una función de la librería antes mencionada desde un programa que él mismo elabore, esto quiere decir que el usuario iniciará su programa que realizará una serie de tareas que son independientes de la plataforma, pero al momento de necesitar entrar en la misma deberá invocar a una función que le permita crear un cliente, suministrándole todos los datos necesarios para ejecutar. Terminada la ejecución el programa podrá continuar con sus diferentes tareas e invocar nuevamente a la plataforma si así lo requiere.

El Inicializador cliente deberá empaquetar los datos a ser enviados al servidor y enviar dichos datos con lo cual solicita al servidor permiso para entrar en la plataforma. Si el acceso es denegado deberá culminar con su trabajo, de lo contrario permanecerá a la espera de un mensaje que le indique que la ejecución ha finalizado con éxito para así comunicárselo al programa del usuario.

IV.2.4.6 Diseño del Proceso Central

El proceso central permanecerá igual que en el primer prototipo salvo que se le incorporará la validación de usuarios, para ello deberá hacer una consulta a la base de datos lo cual le permitirá verificar que el usuario suministrado por el cliente sea válido.

IV.2.4.7 Diseño del Proceso Maestro

A fin de llevar a cabo las pruebas será necesario realizar algunos cambios al proceso maestro, para ello el programa deberá capturar el tiempo del sistema un instante antes de iniciar el procesamiento y un instante después de recibir el mensaje de culminación del mismo. Esta medición se llevará a cabo con el fin de evaluar el desempeño de la plataforma paralela en comparación con una ejecución secuencial.

IV.2.5 Implementación del segundo prototipo

En esta etapa se explicarán las nuevas incorporaciones hechas a la plataforma tales como: librería general, librería de transacciones con la base de datos, validación de usuarios, Inicio Plataforma, Inicializador Cliente e Inicializador Servidor.

IV.2.5.1 Implementación de la librería general

Las librerías de enlace estático son archivos cuya finalidad es almacenar funciones y variables globales. Este tipo de librerías se copia al programa cuando es compilado, por lo cual una vez generado el ejecutable del mismo, la librería puede ser desechada puesto que su código fuente ya no es necesario para la ejecución, debido a que todas la funciones, variables y procedimientos necesarios para ejecutar se han añadido al programa.

Las librerías ofrecen la ventaja de dar más orden al programa desarrollado, evitando la duplicación de códigos utilizados en varios programas o procesos, ofreciendo un mejor performance ya que se ahorra tiempo de compilación.

La librería implementada se denomina libtest.h, dicha librería almacena las declaraciones de las funciones y procedimientos que son comunes para uno o varios procesos de la plataforma. En el siguiente ejemplo se puede observar su estructura:

```
#ifndef _LIBRERIA_1_H
#define _LIBRERIA_1_H

int enviarMensaje(char msje[100],int ptid);
int clteNuevo(int numMaq, char usr[15],char pwd[15], char arch[50]);
int searchParent();
int createFile (char *fname);
int saveFile(char *data, char *fname, int lon);

#endif
```

Adicionalmente fue necesaria la implementación de cada una de las funciones y procedimientos que conforman la librería, lo cual fue realizado en el archivo funciones.c. A continuación se describe cada una de ellas:

- **enviarMensaje:** recibe como parámetros el mensaje a ser enviado y el identificador del proceso que lo va a recibir, devolviendo un entero que indica el estatus del envío. Para realizar su cometido emplea tres funciones de la PVM, la primera de ellas es *pvm_initsend* que indica a la misma que tipo de codificación se va a utilizar para hacer el envío. La segunda es *pvm_pkstr* que empaqueta un mensaje de tipo cadena. La tercera es *pvm_send* que realiza el envío al proceso indicado.
- **clteNuevo:** esta función tiene como objetivo montar al cliente que va a hacer la solicitud del servicio, por tanto es la función que va a invocar el usuario desde su código cuando desee iniciar un procesamiento en la plataforma. Recibe como parámetros el número de máquinas necesarias para ejecutar, el nombre de usuario, contraseña y archivo donde se encuentra la información a procesar.

  Como primer paso empaqueta la información que va a usar el cliente para hacer la solicitud del servicio. Una vez empaquetada, utiliza la función *pvm_spawn* que montará al proceso hijo (Inicializador Cliente) en la máquina desde donde se hace la solicitud. Finalmente permanece a la espera de mensajes de notificación que le indiquen al usuario si la ejecución se llevó a cabo con éxito o si ocurrió alguna falla.

- **searchParent:** esta función es empleada por Inicializador Cliente para ubicar a Inicializador Servidor, esto se debe a que por ser el cliente levantado por el usuario de forma independiente al servidor, no posee una manera automática de conocer el identificador otorgado por la PVM al Inicializador Servidor, por tanto fue necesario elaborar una rutina que le indicara al cliente el identificador del proceso al cual debe enviar los datos del usuario, en este caso al servidor. Para

esto se debió verificar que tareas estaban corriendo en la máquina virtual con es uso de la función *pvm_tasks*.

- **createFile:** tiene como finalidad crear el archivo de texto donde se van a almacenar los logs de la plataforma.
- **saveFile:** es utilizada para escribir en el archivo de texto la información de los logs de la plataforma.

Para compilar la librería se utilizaron los siguientes comandos:

```
gcc libtest.h -o libtest.o
gcc -c funciones.c -o funciones.o
ar rcs libtest.a funciones.o
```

Como se puede observar el fichero de la librería es pasado como argumento a gcc. Se obtiene así el fichero objeto (funciones.o) del archivo fuente (funciones.c).

El comando *ar* es un comando GNU que permite generar archivos de enlace estático (libtest.a). El mismo suele ser utilizado para crear librerías, puesto que agrega una tabla en el archivo con todos los símbolos de los ficheros objeto (funciones.o), que después serían utilizados por el compilador (*gcc*).

IV.2.5.2 Implementación de la librería de la base de datos

Al igual que en el caso anterior se creó una librería estática denominada lidDB.h que posee la siguiente estructura:

```
#ifndef _LIBRERIA_2_H
#define _LIBRERIA_2_H

int iniciarSesion1(char us[10], char pwd[15]);
int consultar ();
int actualizar(int tid);
int cliente(int tid, char dataF[50],int nMaq);

#endif
```

Esta librería será utilizada por los procesos ubicados en el servidor para realizar diferentes transacciones con la base de datos. Debido a que el manejador de bases de datos que se utilizó fue MySQL, se debió incluir en el fichero funcionesDB.c la librería mysql.h. A continuación se explican cada una de las funciones que se encuentran implementadas en dicho fichero:

- **iniciarSesion1:** es la función utilizada por el proceso central para validar al usuario. Para ello emplea tres variables declaradas de la siguiente manera:

```
MYSQL *conn;
MYSQL_RES *res;
MYSQL_ROW row;
```

La primera de ellas será utilizada para realizar la conexión con la base de datos, suministrándole información del nombre de la base de datos, usuario, contraseña, servidor, entre otros, para ello se invoca a la función *mysql_real_connect.*

Una vez conectada se indica al manejador cual será el comando a ejecutar con dicha conexión, para ello se emplea la función *mysql_query.* Hecho esto se procede a ejecutar la consulta con la función *mysql_use_result*, que es almacenada en la variable res. Finalmente se recorren los registros traídos por la consulta almacenados en la variable row con el uso de la función *mysql_fetch_row.*

Finalmente se retorna un valor entero que indica el estatus de la consulta realizada.

- **Consultar:** esta función tiene la finalidad de consultar en la base de datos la información almacenada respecto al proceso Inicio Plataforma, esto con el objetivo de obtener el identificador de dicho proceso, ya que será al que se le envíen los mensajes de estatus de la ejecución puesto que es el encargado de escribirlos en el archivo de logs.
- **Actualizar:** está encargada de actualizar los datos del proceso Inicio Plataforma cada vez que la aplicación sea iniciada, esto se debe a que la PVM otorga identificadores que pueden ser diferentes cada vez que el mismo proceso entre en la máquina virtual.
- **Cliente:** tiene como finalidad almacenar los datos del cliente que hace la solicitud a fin de llevar un control de cada uno de ellos.
- **ConsultarCliente:** esta función tiene la finalidad consultar los datos de los clientes que se encuentran usando la plataforma.
- **EliminarCliente:** tiene como objetivo eliminar los datos del cliente que haya culminado su ejecución.

Para compilar y generar la librería se ejecutaron en el terminal los siguientes comandos:

```
cc libDB.h -o libDB.o
gcc -I/usr/include/mysql -L/usr/lib/mysql -c funcionesDB.c -o funcionesDB.o
```

ar rcs libDB.a funcionesDB.o

Como se pudo observar, fue necesario indicar a *gcc* la ubicación de la interfaz de MySQL así como la ubicación de la librería mysql.h.

IV.2.5.3 Implementación del proceso Inicio Plataforma

Inicialmente se debe indicar a la PVM que un proceso va a entrar a la máquina virtual, para ello se invoca a la función *pvm_mytid*, que alista al proceso para su entrada otorgándole un identificador único dentro de la misma.

Una vez alistado se realiza una llamada al sistema de la siguiente forma: *gethostname(localH, 64)*, con ello se obtiene el nombre de la máquina donde se está ejecutando la plataforma, esto es utilizado para indicar al Inicializador Servidor en qué maquina del cluster se debe iniciar.

Como siguiente paso se realiza una llamada a la función *pvm_spawn*, que está encargada de crear un proceso hijo que se estará ejecutando de forma paralela al padre, en este caso el proceso hijo será Inicializador Servidor.

Una vez levantado el proceso hijo, el padre permanece a la espera de mensajes, para ello se emplea la función *pvm_nrecv*, esta función permite una comunicación asíncrona puesto que el receptor (en este caso el proceso Inicio Plataforma), no debe bloquearse hasta que arribe el mensaje esperado, sino que por el contrario puede continuar realizando otros trabajos. Esta llamada es realizada dentro de un bucle que permanece en constante chequeo del arribo de mensajes, de ser recibido algún mensaje se invoca a la función *pvm_bufinfo*, que devuelve la información del buffer que posee el mensaje y finalmente se procede a desempaquetarlo haciendo una llamada a la función *pvm_upkstr*, la cual desempaqueta un mensaje de tipo cadena.

Paralelamente a esto el proceso deberá hacer uso de las funciones *createFile* y *saveFile* (explicadas anteriormente) con el objetivo de escribir en el archivo de logs los mensajes recibidos.

Cuando el proceso deba abandonar la máquina virtual se invoca a la función *pvm_exit*, que le indica a la PVM que debe eliminarlo de sus registros puesto que ha culminado su ejecución.

IV.2.5.4 Implementación del proceso Inicializador Cliente

Este proceso es activado por el usuario al hacer la llamada a la función *clteNuevo,* cuyo funcionamiento fue explicado anteriormente.

Inicialmente debe invocar a la función *pvm_mytid*, que alista al proceso para su entrada a la máquina virtual otorgándole un identificador único dentro de la misma.

Este proceso posee una comunicación en dos sentidos, una con la función *clteNuevo* que en este caso puede ser vista como el proceso padre y otra comunicación con el Inicializador Servidor, para ello deberá hacer uso de las funciones *pvm_parent* a fin de obtener el identificador del proceso padre y *searchParent* (descrita anteriormente), para obtener el identificador del servidor.

Hecho esto, procede a empaquetar la información de la ejecución a ser enviada al Inicializador Servidor y hace una llamada a la función *enviarMensaje* para hacer la solicitud del servicio al servidor.

Finalmente permanece a la espera de mensajes provenientes de cualquiera de los procesos ejecutándose en el servidor, para ello se emplea la función *pvm_nrecv.* Esta llamada es realizada dentro de un bucle que permanece en constante chequeo del arribo de mensajes, de ser recibido algún mensaje se invoca a las funciones *pvm_bufinfo* y *pvm_upkstr*, descritas anteriormente. El mensaje recibido es enviado al proceso padre (*clteNuevo*) indicándole la información de la ejecución ya culminada. Los mensajes recibidos pueden ser de dos tipos, de error o de éxito en la ejecución, dependiendo de esto, *clteNuevo* deberá devolver al usuario un valor entero indicándole el status.

El proceso culmina así su trabajo y abandona la máquina virtual invocando la función *pvm_exit*, que le indica a la PVM que debe eliminarlo de sus registros puesto que ha terminado su ejecución.

IV.2.5.5 Implementación del proceso Inicializador Servidor

Este proceso es activado por el proceso Inicio Plataforma, está encargado de recibir las solicitudes provenientes de Inicializador Cliente y de proveer el servicio solicitado.

Al igual que en todos los procesos antes descritos, se invoca a la función *pvm_mytid*, que alista al proceso para su entrada a la PVM otorgándole un identificador único dentro de la misma. Así mismo

se invoca a la función *pvm_parent* para obtener el identificador del proceso Inicio Plataforma para el envío de mensajes.

Luego se realiza la llamada al sistema *gethostname(localH, 64)*, con ello se obtiene el nombre de la máquina donde se está ejecutando la plataforma, esto es utilizado para indicar al Proceso Central en qué maquina del cluster se debe iniciar.

Cuando una solicitud arriba, se invoca a las funciones *pvm_nrecv, pvm_bufinfo y pvm_upkstr* (descritas anteriormente), con las cuales se recibe y desempaqueta la solicitud de servicio. En ella se incluye la información del usuario que solicita el servicio así como los datos de la ejecución.

Posteriormente se invoca a la función *validFile,* que está encargada de verificar la validez del archivo de datos, asegurándose que posea el formato adecuado y que se encuentre en la ubicación suministrada. Adicionalmente se invoca la función *cliente,* con el objetivo de almacenar los datos del cliente que ha solicitado el servicio.
Finalmente se emplea la función *pvm_spawn* con la cual se crea al proceso hijo (Proceso Central), al cual se le envía la información necesaria para iniciar la ejecución.

IV.2.5.6 Implementación del Proceso Central

Este proceso conserva todas las funcionalidades descritas en el prototipo anterior pero incorporando la validación de usuarios, para ello se hace una llamada a la función *iniciarSesion1,* la cual forma parte de la librería libDB.h descrita anteriormente y tiene como finalidad autenticar al usuario contra la información almacenada en la base de datos.

Adicionalmente se invoca la función *consultar* que devuelve la información del proceso Inicio Plataforma realizando una consulta a la base de datos, permitiendo así el envío de mensajes hacia ese proceso.

IV.2.5.7 Implementación del Proceso Maestro

Este proceso permanece igual que en el prototipo anterior pero incorporando mediciones del tiempo de ejecución de los procesos esclavos, datos que serán utilizados para probar el desempeño de una ejecución en paralelo en comparación con una secuencial.

Para medir el tiempo de la ejecución se empleó la función *time*, la cual fue invocada un instante antes de enviar la data a las máquinas esclavas y después de recibir el mensaje de culminación de ejecución, permitiendo así capturar la hora en que comenzó el envío (*sysT1*) y la hora a la que finalizó (*sysT2*). Posteriormente se empleó la función *difftime* con la cual se obtiene la diferencia en segundos de los dos tiempos antes mencionados. De esta manera se captura el tiempo de ejecución empleado por la plataforma para devolver los resultados.

IV.2.6 Pruebas

Para las pruebas del segundo prototipo se empleó un mecanismo que permite medir el desempeño dentro del sistema paralelo. Para ello se debieron medir dos tipos de tiempos, el tiempo de ejecución en serie y el tiempo de ejecución en paralelo.

El tiempo de ejecución en serie (t_s) abarca desde el momento en que se inicia la ejecución hasta el momento en que se ejecuta la última línea de código.

Por su parte el tiempo de ejecución en paralelo (t_p) en este caso abarca tanto el tiempo empleado en el procesamiento como el empleado en el envío de mensajes a las máquinas esclavas.

Para mostrar la efectividad de la plataforma APPD se escoge una aplicación basada en una técnica de optimización global propuesta por Buitrago y colaboradores para ajustar los parámetros asociados a un modelo microscópico de tráfico vehicular de comportamiento emergente basado en un autómata celular. (Buitrago, S., Aponte, A., Rodríguez, G. Global optimization techniques and their applications to traffic modeling. Enviado para publicación. 2006).

En esta técnica de optimización aparecen tres aspectos que determinan la eficiencia de la misma: la determinación de los puntos iniciales, un algoritmo de búsqueda propiamente dicho y los criterios de parada. Se escoge el primer aspecto para ser calculado con la plataforma de procesamiento distribuido APPD, dado su naturaleza paralela. La idea detrás del cálculo de los puntos iniciales es cubrir el dominio de búsqueda (dominio donde varían los parámetros mencionados arriba) con un número finito de puntos, de forma de obtener la mayor información posible de la función objetivo.

Las condiciones bajo las que se realizó la simulación fueron las siguientes:

- La red utilizada fue Fast Ethernet (100 Mbits/s).

- Se asumió que ninguno de los procesadores se encontraba ejecutando otro proceso.

El código de prueba antes descrito fue ejecutado inicialmente de forma secuencial sin el uso de la plataforma, dicho programa debió realizar nueve procesamientos diferentes iniciando cada uno de ellos después de haber culminado el anterior, de lo cual se obtuvo un tiempo secuencial igual a: $t_s = 119seg$

Para las mediciones en paralelo se empleó el mismo código de prueba pero con la incorporación de algunos cambios a fin de poder utilizarlo dentro de la plataforma. Para ello se construyó dentro de dicho programa de prueba, el archivo de datos que sería utilizado por APPD para iniciar el procesamiento, adicionalmente se invocó la función *cltenuevo* contenida en la librería libtest.h (antes descrita), con el objetivo de solicitar el servicio de ejecución creando un cliente nuevo.

Con este fin se llevaron a cabo tres mediciones diferentes, la primera de ellas empleando sólo una máquina del cluster para el procesamiento, de esta ejecución se obtuvo un tiempo paralelo igual a: $t_p = 159seg$

Como se puede observar el tiempo de ejecución empleado por la plataforma con una sola máquina es superior al tiempo secuencial, esto se debe a que el Proceso Maestro debe llevar a cabo una serie de tareas a fin monitorear las ejecuciones remotas, para lo cual emplea un tiempo considerable que marca la diferencia si se realiza la ejecución con una máquina. Esta prueba a pesar de ser hecha con la plataforma puede ser considerada secuencial puesto que APPD procesa una tarea por máquina esclava a la vez.

Posteriormente se realizó una segunda prueba esta vez empleando dos máquinas para la ejecución, obteniéndose como resultado: $t_p = 61seg$, con este resultado se observa una disminución considerable en el tiempo de ejecución.

Finalmente se llevó a cabo una tercera prueba empleando para ello tres máquinas, con lo cual se obtuvo $t_p = 41seg$.

Con esta prueba fue posible observar el comportamiento de la plataforma ante tres escenarios diferentes y mostrar así en qué casos es recomendable su uso y en cuales no resulta favorable.

CAPÍTULO V
RESULTADOS

La metodología del Prototipo Evolutivo permitió cumplir con éxito los objetivos planteados para el presente trabajo especial de grado, obteniéndose así los siguientes resultados:

- De las entrevistas realizadas al tutor se determinaron aspectos importantes a tener en cuenta para el desarrollo, tales como: tipo de cluster (Cluster Beowulf), características de la plataforma (disponibilidad y transparencia), ambientes de trabajo tanto para el cluster (sistema operativo Fedora Core 4, protocolo para compartir archivos remotos (NFS) y administración centralizada de cuentas de usuarios (NIS), así como para la plataforma se emplearía el lenguaje de programación C por permitir implementaciones óptimas y eficientes. Y finalmente el uso de una herramienta de paso de mensajes para la comunicación entre procesos.

- Se realizó una investigación con base teórica en la cual se analizaron las características más relevantes de las dos librerías de paso de mensajes por excelencia, PVM (Parallel Virtual Machine) y MPI (Message Passing Interface). De dicha investigación se seleccionó el paquete de la PVM como herramienta a utilizar para el paso de mensajes puesto que ofrece mecanismos de control de recursos dinámicamente y permite la elaboración de aplicaciones tolerantes a fallos.

Con base en la investigación teórica antes mencionada:

- Se diseñó la arquitectura del cluster basada en el tipo cliente - servidor, cuya comunicación entre las máquinas se lleva a cabo bajo el protocolo TCP/IP, contando con recursos como NIS, NFS y PVM para cumplir su cometido.

- Se diseñó la arquitectura de la plataforma basada en el tipo cliente – servidor, donde existe un proceso maestro encargado de distribuir la data y enviar los resultados y un proceso esclavo encargado de procesar la data recibida del maestro y devolverle los resultados al mismo.

- Para el cumplimiento del objetivo general de este trabajo especial de grado se debió realizar el montaje del cluster y la implementación de la plataforma de procesamiento paralelo distribuido:

1. Como resultado de las configuraciones de servicios y protocolos para el montaje del cluster se obtuvo lo siguiente:

   a) Servidor NFS, encargado de administrar las máquinas que tienen acceso a las carpetas compartidas.
   b) Servidor NIS, que provee una base de datos centralizada de usuarios del cluster.
   c) Cliente NFS, encargado de montar el fichero remoto en la máquina cliente.
   d) Cliente NIS, encargado de localizar al maestro que ofrece el servicio NIS para así poder solicitar información de usuarios al servidor.
   e) PVM, que permite montar una máquina virtual paralela que se basa en el paso de mensajes para comunicar procesos.

2. Como valor agregado a la implementación de la plataforma se añadieron dos librerías estáticas con las funciones comunes a los procesos a fin de evitar código duplicado. Adicionalmente se incorporaron dos procesos a la plataforma (Inicializador Cliente e Inicializador Servidor), esto con el fin de proveer mayor transparencia al usuario al acceder a la plataforma. De acuerdo a esto se obtuvo lo siguiente:

   a) Dos librerías estáticas que poseen las funciones a utilizar por los procesos para llevar a cabo su cometido.
   b) Proceso Inicio Plataforma, reside en el servidor y está encargado de iniciar el ambiente activando al inicializador servidor, permaneciendo así a la espera de los mensajes provenientes del resto de los procesos para mostrarlos por pantalla y generar el archivo de logs.
   c) Proceso Inicializador Servidor, el cual reside en el servidor y está encargado de esperar solicitudes de ejecución provenientes de los clientes, de verificar la validez del archivo de datos suministrado y de dar inicio al Proceso Central.
   d) Proceso Inicializador Cliente encargado de empaquetar y enviar los datos del usuario al servidor con lo cual realiza la solicitud de servicio al mismo. Permanece a la espera de un mensaje que le indique que la ejecución ha finalizado para dar fin a su trabajo. Este proceso reside en las máquinas clientes y es iniciado dentro del programa que el usuario esté ejecutando en el momento en que requiera del servicio de la plataforma.

e) Proceso Central encargado de autenticar al usuario que desee acceder a la aplicación y verificar la disponibilidad de máquinas para la ejecución. Este proceso reside en el servidor y tiene como labor autorizar la ejecución y dar inicio al Proceso Maestro.

f) Proceso Maestro, encargado de distribuir la data entre las máquinas disponibles, monitorear la ejecución de las tareas, reasignar la data en caso de fallos y terminar la ejecución del usuario. Este proceso reside en el servidor y tiene la labor de activar a los Procesos Esclavos.

g) Proceso Esclavo, cuya función se basa en procesar la data que le fue asignada y devolver los resultados. Este proceso reside en las máquinas esclavas o clientes y su labor final es comunicar a la plataforma cuando haya terminado su ejecución.

- Como resultado de las pruebas se obtuvo lo siguiente:

1. Se probó el buen funcionamiento de los protocolos, servicios y paquetes configurados para el cluster, pudiendo así compartir archivos y accederlos remotamente (NFS), tener acceso remoto a cuentas de usuarios (NIS) y comunicación de procesos remotos a través del paso de mensajes (PVM).
2. Se realizaron pruebas basadas en el criterio de caja negra, permitiendo esto observar que la plataforma cumpliera con sus objetivos y determinar posibles errores no detectados anteriormente. Finalmente se empleó un mecanismo que permitió medir el rendimiento del sistema paralelo. Para ello se utilizó un aplicación basada en una técnica de optimización global para ajustar los parámetros asociados a un modelo microscópico de tráfico, con el que se realizaron mediciones del tiempo de su ejecución en serie y con el uso de la plataforma, a fin de comparar ambos casos. Esta prueba permitió observar el comportamiento de la plataforma de acuerdo a las condiciones de la ejecución, mostrando en cuales casos es conveniente su uso y en cuales no es favorable.

CAPÍTULO VI
CONCLUSIONES Y RECOMENDACIONES

VI.1 Conclusiones

Tomando como base los resultados obtenidos durante el desarrollo del presente trabajo especial de grado, se obtuvieron las siguientes conclusiones:

1. Los clusters ofrecen capacidades de cálculo superiores a las de un ordenador, siendo capaces de brindar un alto rendimiento, alta disponibilidad, equilibrio de carga o cómputo adaptativo, fáciles de integrar a redes existentes, escalabilidad, pudiendo adaptarse a un número de usuarios cada vez mayor, sin perder calidad en los servicios.

2. Las librerías de paso de mensajes son herramientas de gran importancia para la programación concurrente puesto son capaces de aportar sincronización entre procesos. Su principal característica es que no requiere de memoria compartida, lo cual las hace primordiales en la programación para sistemas distribuidos dado que no existen recursos directamente compartidos para intercambiar información entre los procesos.

3. La PVM ofrece disponibilidad, tolerancia a fallos, monitoreo dinámico de recursos del sistema, portabilidad, heterogeneidad permitiendo abstraer la topología de la red, la tecnología de la red, la cantidad de memoria de cada máquina, el tipo de procesador y la forma de almacenar los datos. Paralelismo escalable, permitiendo definir cuantos procesadores puede utilizar una aplicación y en caso de que falten, realiza el trabajo en menos procesadores, recurriendo a técnicas de procesamiento concurrente.

4. La plataforma desarrollada (APPD) ofrece al usuario transparencia de acceso y de ubicación puesto que puede realizar sus operaciones sin importar su ubicación física. Transparencia de migración o movilidad debido a que es capaz de reubicar datos sin afectar la ejecución del usuario. Transparencia de paralelismo puesto que las actividades pueden ejecutarse en paralelo sin el conocimiento del usuario.

5. APPD brinda alta disponibilidad puesto que es capaz de detectar fallas y continuar con su trabajo sin afectar la ejecución.

6. APPD es de fácil escalabilidad puesto que requiere de pocos cambios a la hora incorporar un número mayor de recursos de hardware al cluster, tales como servidores esclavos o nuevos clientes.

7. El procesamiento de data bajo el ambiente de la plataforma ofrece un ahorro considerable de tiempo puesto que el usuario puede iniciar la ejecución sin necesidad de estar en constante chequeo de la misma, siendo responsable la plataforma de culminar exitosamente el procesamiento, aun cuando ocurran fallas durante la ejecución. Adicionalmente ofrece la posibilidad de ejecutar simultáneamente o en forma paralela procesos de alto desempeño, que en otras situaciones debían ser ejecutados en forma secuencial, puesto que se dispone de un grupo de esclavos que permiten elevar las capacidades de cómputo, un mayor aprovechamiento de los recursos disponibles y disminuir los tiempos de espera. Sin embargo se debe tener en cuenta que existen factores limitantes que pueden afectar los tiempos de respuesta, como por ejemplo emplear una sola máquina para la ejecución o realizar un procesamiento secuencial bajo el ambiente de la plataforma, puesto que ésta debe realizar una serie de tareas para hacer la ejecución remota, representando una pérdida de tiempo innecesaria en un procesamiento secuencial.

VI.2 Recomendaciones

1. Incorporar un número considerable de hosts al proyecto a fin poder aprovechar las bondades de la PVM y de la plataforma de procesamiento paralelo distribuido.

2. El paradigma de programación funcional puede presentar cuellos de botella, es posible mejorar su escalabilidad pasando de un procesador maestro (servidor) a un conjunto de procesadores maestros (conjunto de servidores esclavos) controlando estos a su vez a un conjunto de procesadores esclavos (máquinas clientes). Para esto será necesario realizar algunos cambios en la plataforma a fin de que procesos como el maestro sean ejecutados no en un servidor sino en varios, mejorando así el desempeño. Sin embargo esta alternativa puede acarrear algunas consecuencias negativas puesto que en caso de falla de alguna de las máquinas maestro, ésta

sería excluida de la PVM y por tanto la ejecución quedaría inconclusa. Al tomar esta decisión se debe tener presente que se sacrifica en cierto nivel la recuperación de fallas por el desempeño.

3. A fin de disminuir las colisiones y aumentar la velocidad se recomienda incluir dos switch en lugar de uno solo.

4. Agregar un módulo de administración de cuentas de usuarios autorizados a utilizar la plataforma, esto con el fin de poder agregar nuevos usuario o eliminar existentes, cambio de claves, entre otros.

5. Agregar un módulo cliente que permita al usuario suministrar de forma más segura sus datos, permitiéndole introducir en tiempo real su nombre de usuario y contraseña y la información de la ejecución, haciendo la autenticación no en el proceso central sino antes de iniciar cualquier solicitud, ya que actualmente este tipo de información se encuentra en el código fuente del usuario y es validado después de hacer la solicitud de servicio.

BIBLIOGRAFÍA

Libros

- Sterling, T. (2001) Beowulf Cluster Computing with Linux. Cambridge: MIT Press.
- Slern, H., Eisler, M., Labiaga, R. (2001) Managing NFS and NIS. Cambridge: O'Reilly Media, Incorporated.
- Brown, R. (2004) Engeneering a Beowulf-style Compute Cluster. Durham: Duke University Physics Department.
- Geist, A., Beguelin, A., Dongarra, J., Jiang, W. Manchek, R., Sunderam, V. (2000) PVM Parallel Virtual Machine. A user's guide and tutorial for networked parallel computing. Massachusetts: Massachusetts Institute of Technology.
- Sterling, T. Salmon, J., Becker, D., Savarese, D. (1999) How to build a Beowulf. A Guide to the Implementation and Application of PC Clusters. Massachusetts Institute of Technology.
- Snir, M., Huss-Lederman, S., Walker, S. Dongarra, J. (1999) MPI—The Complete Reference Volume 1, The MPI Core. Massachusetts: Massachusetts Institute of Technology.
- Dormido, S. Hernandez, R., Ros, S., Sánchez, J. (2003) Procesamiento Paralelo. Madrid: Editorial Sanz y Torres, S.L.
- Tanenbaum, A. (1996) Sistemas Operativos Distribuidos. Méjico: Prentice Hall Hispanoamericana S.A.
- El-Rewini, H., Abd-El-Barr, M. (2005). Advanced Computer Architecture and Parallel Procesing. New Jersey: Wiley & Sons, Inc.

Documentos Electrónicos

- Lam – MPI (2006). Top 10 reasons to prefer MPI over PVM [en línea]. Disponible en: http://www.lam-mpi.org/mpi/mpi_top10.php. [2006, Abril].

Artículos Electrónicos

- Apon, A., Buyya, R., Jin, H., Mache, J. (2001). Cluster Computing in the Classroom: Topics, Guidelines, and Experiences.

- Pino, O., Arroyo, R., Nievas F. (2002). Los Clusters como plataforma de procesamiento paralelo.
- Alonso, J. (1997). Programación de aplicaciones paralelas con MPI (Message Passing Interface).
- Gropp, W., Lusk, E. (2002). Goals Guiding Disign: PVM and MPI.
- Lage1, F., Zubenko, Y., Cataldi, Z. (2001). An Extended Methodology for Educational Software Design: Some critical points.
- Calderón, A., García, F., Carretero, J. (1999) Una implementación multithread de MPI para LINUX [en línea]. Disponible en: http://es.tldp.org/Presentaciones/1999hispalinux/conf-acaldero/3.pdf, [2005, Diciembre].
- López, J. (2005) [en línea]. Protocolo SSH. Ampliación de Sistemas Operativos. Disponible en: http://sopa.dis.ulpgc.es/ii-aso/portal_aso/leclinux/seguridad/ssh/ssh.pdf, [2006, Abril].
- Red Hat, Inc. (2001). Oficial Red Hat Linux Referente Guide [en línea]. Disponible en: http://www.europe.redhat.com/documentation/rhl7.1/rhl-rg-es-7.1/index.php3, [2006, Abril].
- Castro, J., Cárdenas, L., Parra, I. (1999). Tutorial de Sistemas Distribuidos [en línea]. Disponible en: http://www.itlp.edu.mx/publica/tutoriales/sistsdist1/, [2005, Diciembre].
- Barney, B. (2006). Message Passing Interface (MPI) [en línea]. Disponible en: http://www.llnl.gov/computing/tutorials/mpi/ , [2006, Febrero].

Páginas Web

- Wikipedia, la enciclopedia libre que todos podemos editar. Disponible en: http://www.wikipedia.com
- Fedora. Disponible en : http://fedora.redhat.com/
- PVM: Parallel Virtual Machine. Disponible en: http://www.csm.ornl.gov/pvm/

GLOSARIO

A

API (Application Programming Interface): interfaz de programación de aplicaciones mediante la cual los programas invocan servicios del sistema operativo.

Arquitectura: Término que se refiere a la estructura general de un procesador, sistema operativo, un ordenador o cualquier otro elemento

B

Base de datos: Sistema de almacenamiento de datos muy flexible que permite utilizar la información en función de diversos criterios.

Buffer: Espacio de la memoria que se utiliza como regulador y sistema de almacenamiento intermedio entre los dispositivos de un equipo.

C

Cifrado: es el tratamiento de los datos contenidos en un paquete a fin de impedir que nadie excepto su destinatario pueda leerlos. Constituyen la base de la seguridad de la red.

Cliente: Un sistema o proceso que solicita a otro sistema o proceso que le preste un servicio. Una estación de trabajo que solicita el contenido de un fichero a un servidor de ficheros es un cliente de este servidor.

CPU: Central Prosessing Unit. Unidad Central de proceso de información en un ordenador, donde se realizan las funciones lógicas y matemáticas

D

Directorio: Espacio lógico de una estructura jerárquica en forma de árbol que contiene la información almacenada en un ordenador, habitualmente contenida en ficheros. Un directorio se identifica mediante un nombre, por ejemplo "Mis documentos".

E

Ejecutable: archivo que contiene las instrucciones necesarias para su puesta en marcha.

Escalabilidad: Característica que permite crecer a un PC, una red una aplicación en potencia, tamaño o prestaciones.

Estación: Conjunto formado por un ordenador y sus periféricos dentro de una red.

H

Host: Ordenador que permite a los usuarios comunicarse con otros sistemas centrales de una red. Los usuarios se comunican utilizando programas de aplicación, tales como el correo electrónico, Telnet y FTP.

I

Internet: conjunto de interconexiones formado por todas las redes que utilizan TCP/IP, incluido a partir del ARPANET de finales de 1960 y principios de 1970. Internet conecta aproximadamente 70.000 redes independientes formando una amplia red global.

K

Kernel: Núcleo básico del sistema operativo, a partir del cual se establecen las distintas capas para su integración con el hardware, para la entrada y salida de datos, etc.

L

Linux: es un Sistema Operativo basado en un Kernel Unix, multi-usuario y multi-área. Fue creado en 1991 por Linus Torvalds.

M

Maestro: Dícese del disco que funciona como principal en aquellos ordenadores que utilizan más de un disco duro. Los secundarios reciben el nombre de " esclavos".

Máscara de red: combinación de bits que sirve para delimitar el ámbito de una red de computadores.

N

Network: (red) Una red de ordenadores es un sistema de comunicación de datos que conecta entre sí sistemas informáticos situados en diferentes lugares. Puede estar compuesta por diferentes combinaciones de diversos tipos de redes.

Nodo: Por definición punto donde convergen más de dos líneas. A veces se refiere a una única máquina en Internet. Normalmente se refiere a un punto de confluencia en una red.

L

LAN: (Red de Área Local) Red de datos para dar servicio a un área geográfica máxima de unos pocos kilómetros cuadrados, por lo cual pueden mejorar los protocolos de señal de la red para llegar a velocidades de transmisión de hasta 100 Mbps (100 millones de bits por segundo).

O

Optimización: Modificar el software o el hardware para mejorar los resultados.

P

Paquete: Cantidad mínima de datos que se transmite en una red o entre dispositivos. Tiene una estructura y longitud distinta según el protocolo al que pertenezca. También llamado trama.

Protocolos: Descripción formal de formatos de mensaje y de reglas que dos ordenadores deben seguir para intercambiar dichos mensajes.

Portabilidad: esfuerzo necesario para transferir el programa de un entorno, hardware o software, a otro entorno diferente.

S

Servidor: todo computador que permite a otro conectarse a él mediante un programa cliente, que provee servicios para compartir información y/o recursos.

Sistema: es un conjunto de partes que trabajan relacionadas para cumplir un objetivo común.

Software: conjunto de programas procesados, reglados y documentación; relativos al funcionamiento de un grupo de tratamientos de información.

APÉNDICES

Apéndice A – Clusters

A.1 Tipos de Clusters:

- **Escasamente acoplados:** Una agrupación de computadores está escasamente acoplada si, aún siendo capaz de realizar procesamiento paralelo mediante librerías de paso de mensajes o de memoria compartida, no posee un sistema de instalación y gestión integrado que posibilite una recuperación rápida ante fallos y una gestión centralizada que ahorre tiempo al administrador.

- **Medianamente acoplados:** incorporan un sistema centralizado de instalación y gestión. Esto posibilitaría una rápida intervención del administrador en caso de error en alguno de los nodos, debido además a que la instalación del sistema operativo de cada uno de los nodos de cómputo es considerada temporal y nunca se almacenan datos relevantes para la estabilidad o coherencia del sistema. Esta nueva aproximación representa un paso interesante en la evolución de los clusters, debido a que permite una gran escalabilidad del sistema y posibilita la inclusión de diferentes paquetes de software dentro del propio sistema de gestión e instalación para su instalación automática.

- **Altamente acoplados:** implica eliminar las unidades de disco de los nodos de cómputo para así utilizar únicamente el procesador y la memoria de cada uno de ellos.

A.2 Arquitectura de un cluster:

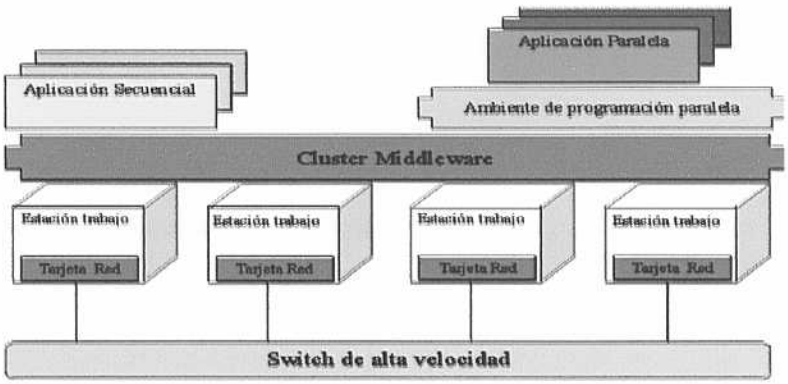

Figura A.1 Arquitectura de un cluster
Fuente: Pino Morillas, Oscar (2002).

Los principales componentes de un Cluster son: red de comunicaciones, estaciones de trabajo, tarjetas de interfaz de red, sistema operativo especial, protocolos y servicios para una comunicación rápida, middleware (hardware, Kernel del sistema operativo, aplicaciones y subsistemas), entorno de programación paralela (Parallel Virtual Machine y Message Passing Interface), aplicaciones secuenciales y paralelas.

Middleware:

Interfaz entre las aplicaciones y el hardware del cluster y el sistema operativo. Cada componente se encuentra en distintos niveles o capas:

- SSI (Single System Image): Es una especie de ilusión creada por el hardware o software que hace ver al cluster como una sola máquina ante los usuarios, las aplicaciones y la red. Aporta transparencia, es decir, cada nodo puede acceder a un periférico o disco sin necesidad de conocer su ubicación.
- SAI (System Availability Infrastructure): Proporciona al cluster servicios tales como: recuperación de fallas y tolerancia a fallas.

Apéndice B – Sistemas Distribuidos

B.1 Tipos de transparencia de los Sistemas Distribuidos

Algunos de los tipos de transparencia que se observan en un sistema distribuido se encuentran en la siguiente tabla:

Tipo	Significado
Transparencia de acceso	Utiliza las mismas operaciones sin importar si es de forma remota o local.
Transparencia de ubicación	El usuario no debe saber la ubicación de los recursos, por lo tanto no debe indicarla.
Transparencia de migración o movilidad	Reubicar clientes y recursos sin afectar operación de usuarios y programas.
Transparencia de replica	Tener múltiples ejemplares de cada recurso para usar el disponible (sin intervención del usuario).
Transparencia de concurrencia	Varios usuarios pueden compartir recursos de manera automática. Los procesos trabajan concurrentemente sobre recursos compartidos.
Transparencia de paralelismo	Las actividades pueden ocurrir en paralelo sin el conocimiento de los usuarios.

Tabla B.1 Tipos de transparencia en sistemas distribuidos

Fuente: Modificado por el autor (http://extension.info.unlp.edu.ar/so/files/SOD.doc, 2003)

B.2 Tipos de Arquitecturas

La arquitectura define la forma en que los componentes de un sistema interactúan unos con otros y se vinculan con la red subyacente. A continuación se muestran dos de los tipos de arquitecturas más usadas:

- **Arquitectura Cliente/Servidor**: Su principal objetivo es estructurar el sistema operativo como un grupo de procesos en cooperación, denominado servidor y que a su vez ofrezcan servicios a los usuarios denominados clientes.

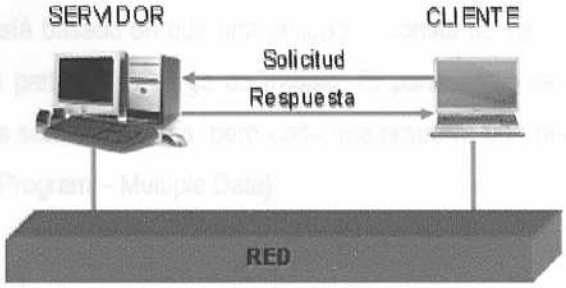

Figura B.1 Arquitectura Cliente/Servidor.
Fuente: Elaborado por el autor.

- **Arquitectura Proxy o Caché**: caché es un almacenamiento de objetos de datos que se usaron recientemente y que se encuentra más próximo que los objetos en sí. Cuando un proceso cliente necesita un objeto, el servicio caché comprueba en el caché, de conseguirlo le da una copia actualizada, de los contrario lo busca. Un caché puede ubicarse en cada cliente o en un servidor Proxy compartido por varios clientes.

Figura B.2 Arquitectura Proxy o Caché.
Fuente: Elaborado por el autor.

Apéndice C – Parallel Virtual Machine

C.1 Modelo Computacional de la PVM

El modelo de la PVM está basado en que una aplicación consta de varias tareas, donde cada una es responsable de una parte de la carga de trabajo. El paralelismo de data es un método muy común donde las tareas son las mismas, pero cada una resuelve una pequeña porción de la data (SIMD o SPMD, Single Program – Multiple Data).

C.2 Estructuras de las aplicaciones de la PVM

C.2.1 Estructura Supervisor – Trabajadores

En esta estructura existe sólo un nivel de jerarquía, el supervisor y los trabajadores. El supervisor o maestro es iniciado manualmente en una de las máquinas, posee cierto número de responsabilidades o tareas, entre las cuales está asignar trabajo a los trabajadores y recolectar los resultados que estos arrojan. Por su parte los trabajadores o esclavos son activados por el maestro para realizar los cálculos, pueden o no ser independientes, es decir, ser dependientes implica que deban comunicarse con los otros trabajadores antes de devolver el resultado.

C.2.2 Estructura Jerárquica

Permite a los trabajadores crear nuevos niveles de trabajadores. Poseen un maestro encargado de iniciar una serie de esclavos quienes a su vez pueden crear otros niveles de esclavos. Esta estructura puede continuar por varios niveles formando así un árbol de tareas.

C.3 Mensajes

Los mensajes en la PVM pueden ser de cualquier tamaño y contener cualquier formato de data. El remitente no espera por un reconocimiento del receptor, sino que continua tan pronto como el mensaje se haya enviado a la red y el buffer del mensaje pueda ser eliminado o reutilizado con seguridad. Los mensajes son recibidos en el mismo orden en que fueron enviados, el receptor puede bloquearse hasta recibir el mensaje esperado o puede continuar trabajando, incluso es

posible determinar un tiempo de espera para el arribo de un mensaje y notificar si este no llegó en el tiempo fijado.

La librería de la PVM proporciona funciones para empaquetar todos los tipos primitivos de data en un mensaje, en uno varios formatos de codificación. Hay cinco sistemas de codificación y decodificación, por tanto cada buffer de mensaje debe tener un sistema asociado a él.

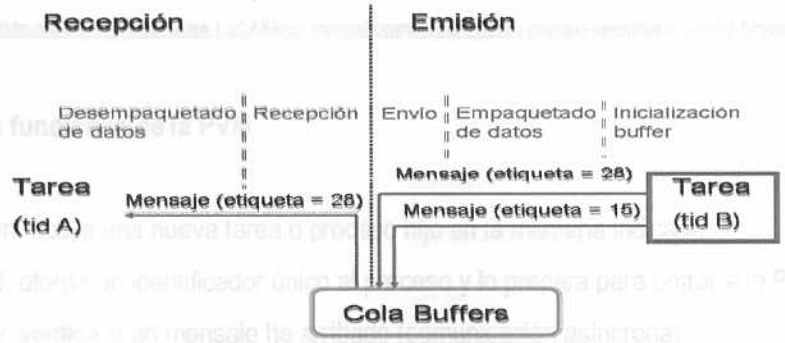

Figura C.1 Envío y recepción de mensajes en la PVM

Fuente: http://asds.dacya.ucm.es/nacho/pp_archivos/Grupo6.ppt

C.3.1 Comunicación Pvmd-Pvmd

Los demonios de la PVM se comunican entre si a través de sockets UDP (User datagram protocol). UDP es un protocolo que implementa un nivel de transporte basado en el intercambio de datagramas que se encapsulan dentro de la parte de datos de un datagrama IP. Este servicio puede duplicar y reordenar los paquetes. Las razones por las que la PVM utiliza sockets UDP se debe a que estos pueden comunicarse con cualquier cantidad de sockets UPD remotos, puede ser inicializado sin comunicación, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera y finalmente porque facilita la tolerancia a fallos puesto que el sistema detecta cuando algún pvmd remoto ha fallado.

C.3.2 Comunicación Pvmd - Tarea y Tarea - Tarea

Las tareas de la PVM se comunican con su Pvmd y con otras tareas a través de sockets TCP (Protocolo de control de transmisión) debido a que entrega los datos de forma confiable, garantizando que estos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron. TCP es la capa intermedia entre el protocolo de Internet (IP) y la aplicación.

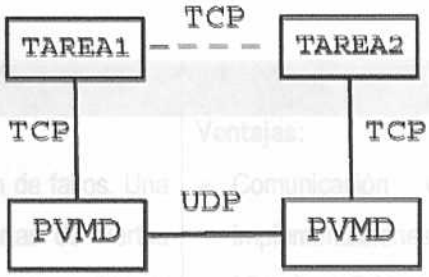

Figura C.2 Protocolos de Comunicación

Fuente: http://es.tldp.org/almacen/Manuales-LuCAS/doc-manual-openMosix-1.0/doc-manual-openMosix_html-1.0/node18_ct.html, 2004

C.3.3 Algunas funciones de la PVM

- pvm_spawn: activa una nueva tarea o proceso hijo en la máquina indicada.
- pvm_mytid: otorga un identificador único al proceso y lo prepara para entrar a la PVM.
- pvm_nrecv: verifica si un mensaje he arribado (comunicación asíncrona).
- pvm_bufinfo: devuelve información del buffer que contiene el mensaje.
- pvm_upkstr: desempaqueta un mensaje de tipo cadena.
- pvm_initsend: limpia el buffer de envío y determinar el tipo de codificación a utilizar en la transmisión del mensaje.
- pvm_pkstr: empaqueta un mensaje de tipo cadena.
- pvm_send: envía la data en el buffer activo.
- pvm_tasks: retorna información de las tareas corriendo en la PVM.
- pvm_tidtohost: retorna el identificador del demonio corriendo la una determinada tarea.
- pvm_config: devuelve la configuración de la máquina virtual.
- pvm_notify: solicita notificación de eventos ocurridos en la PVM.
- pvm_mstat: devuelve el estatus del host solicitado.
- pvm_exit: indica a la máquina virtual que un proceso ha culminado su ejecución.
- pvm_parent: devuelve el identificador del proceso padre.

Apéndice D – PVM vs. MPI

PVM	MPI
Ventajas: • Mecanismo de notificación de fallos. Una tarea pueda manejar alertas de ciertos procesos de los que espera recibir un mensaje. • Monitoreo dinámico de recursos del sistema. • Recursos computacionales heterogéneos manejados como un computador paralelo. • Paralelismo escalable, permite definir cuantos procesadores puede utilizar una aplicación y en caso de que falten, realiza el trabajo en menos procesadores, recurriendo a técnicas de procesamiento concurrente. • Comunicación asíncrona. • Comunicación entre procesos con implementaciones distintas. **Desventajas:** • Una vez que un proceso empieza en una determinada máquina seguirá en ella hasta que se muera. • Comunicación más lenta en la red. • Sacrifica eficiencia en favor de la flexibilidad para comunicar diferentes arquitecturas.	**Ventajas:** • Comunicación entre procesos con implementaciones distintas. • Maneja eficientemente los buffers de mensajes. • Ofrece alta velocidad de comunicaciones en redes. • Amplio conjunto de rutinas de comunicación punto a punto. • Es un estándar. • Es completamente portable. • Comunicación asíncrona. **Desventajas:** • No ofrece mecanismos para la tolerancia de fallos. • Asume un número de procesos estáticos que se debe indicar al iniciar la aplicación. • No es posible determinar dinámicamente los recursos del sistema. • La portabilidad que ofrece es tan amplia que se limitan sus capacidades en otros aspectos y funciones.

Tabla D.1 Ventajas y Desventajas de PVM y MPI

Fuente: Elaboración propia

Apéndice E – Diagramas de Flujo de Procesos del Primer Prototipo

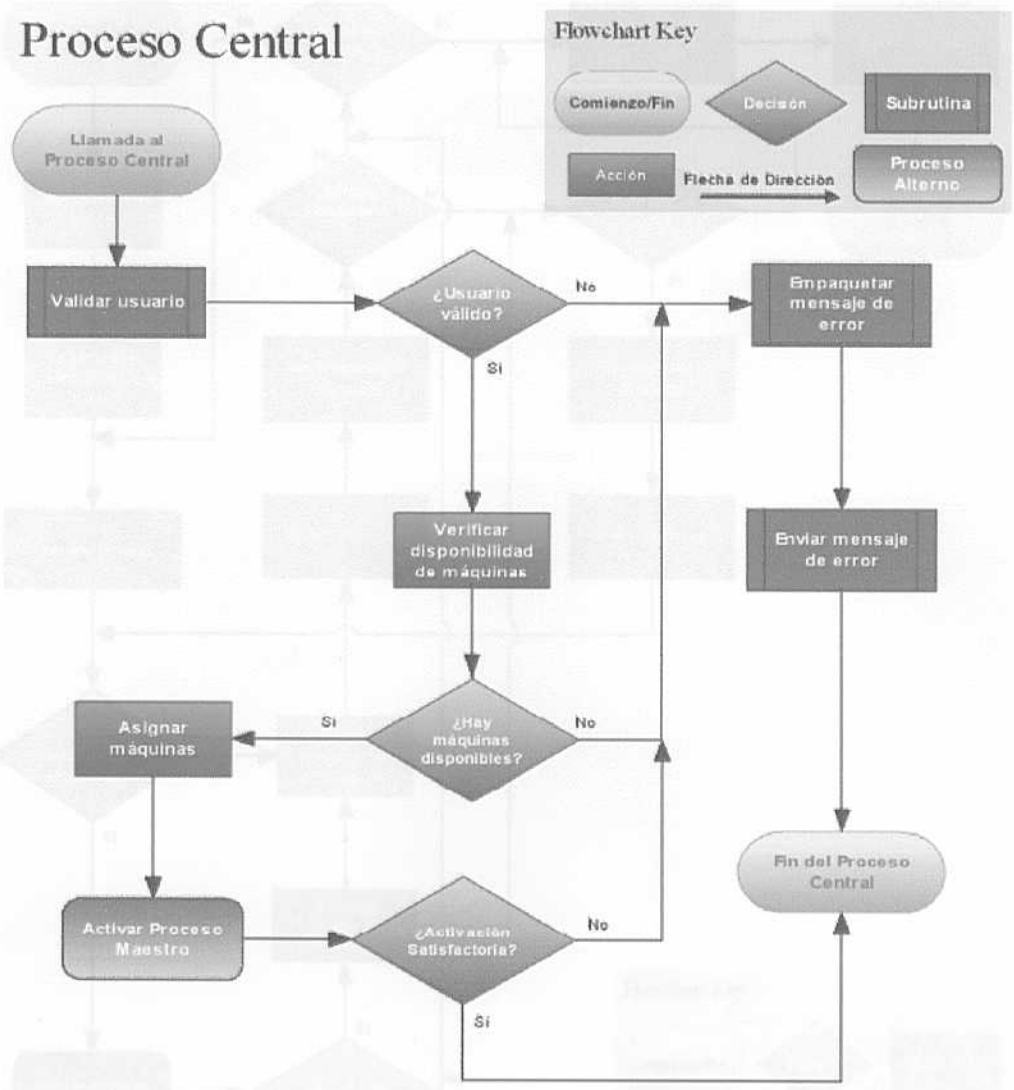

Figura E.1 Diagrama de flujo de procesos del "Proceso Central"

Fuente: Elaboración propia

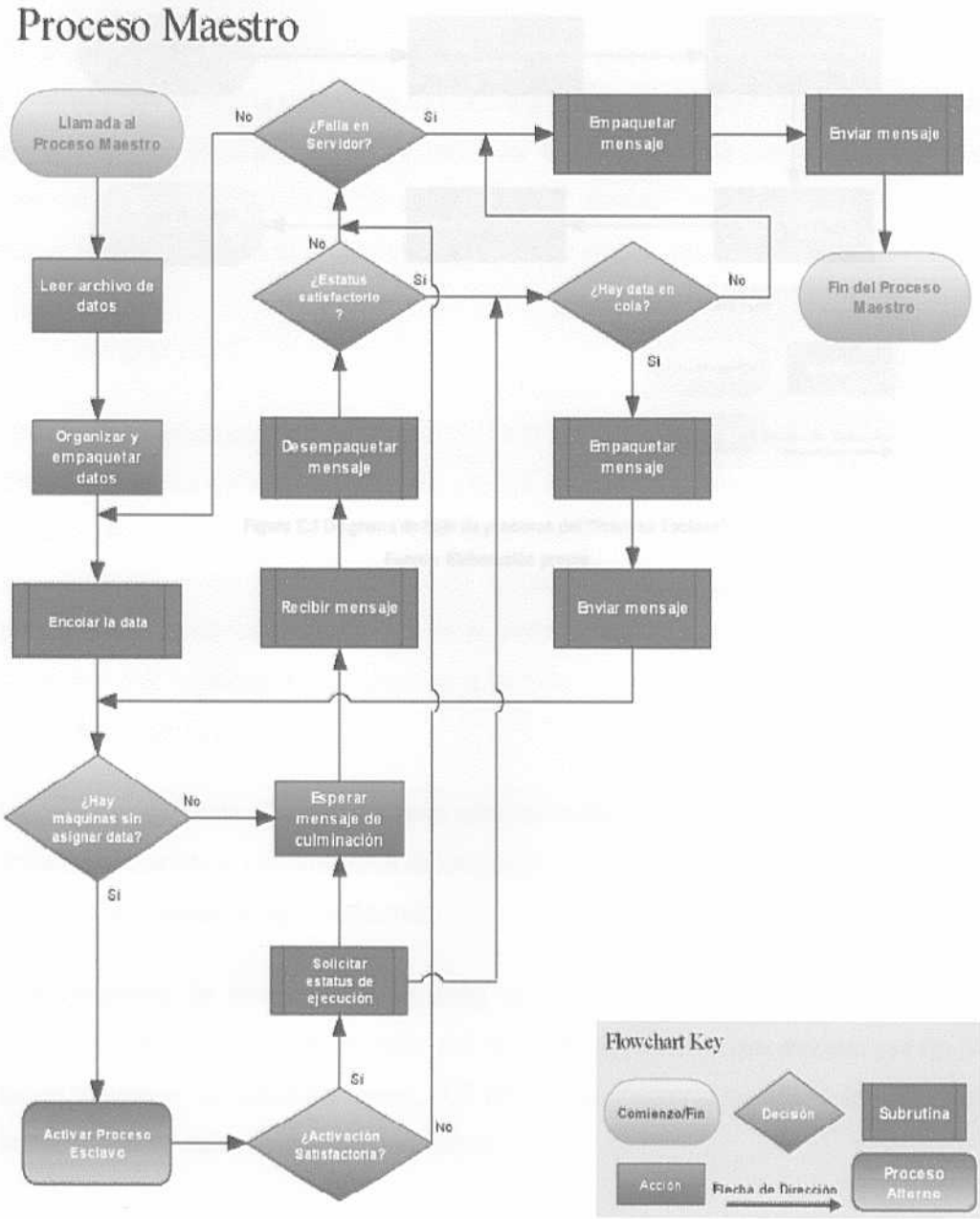

Figura E.2 Diagrama de flujo de procesos del "Proceso Maestro"

Fuente: Elaboración propia

Proceso Esclavo

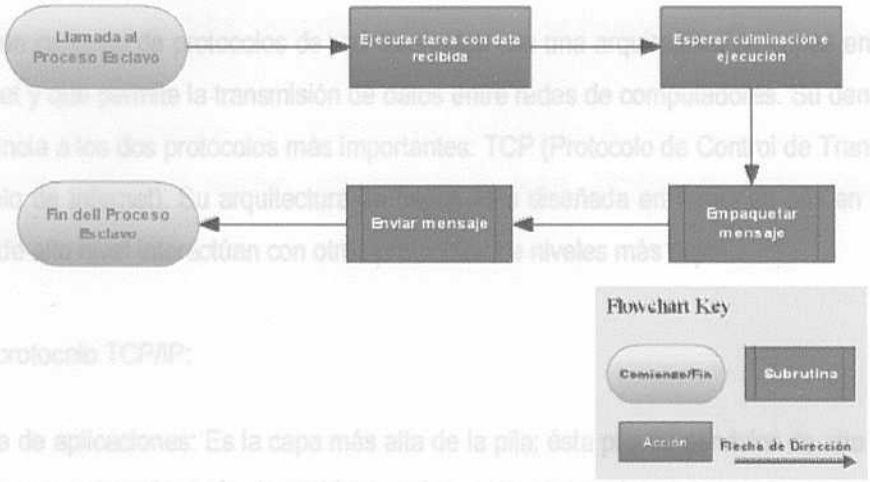

Figura E.3 Diagrama de flujo de procesos del "Proceso Esclavo"

Fuente: Elaboración propia

Apéndice F – Protocolo TCP/IP

TCP/IP es un conjunto de protocolos de red diseñados con una arquitectura en capas en la que se basa Internet y que permite la transmisión de datos entre redes de computadores. Su denominación hace referencia a los dos protocolos más importantes: TCP (Protocolo de Control de Transmisión) e IP (Protocolo de Internet). Su arquitectura en capas está diseñada en forma de pila en la que los protocolos de alto nivel interactúan con otros protocolos de niveles más bajos.

Capas del protocolo TCP/IP:

- La capa de aplicaciones: Es la capa más alta de la pila; ésta provee servicios de alto nivel a los usuarios como transferencia de archivos, entrega de correo electrónico y acceso a terminales remotas, entre otras. Los procesos que se llevan a cabo en este nivel son aplicaciones específicas que pasan los datos al nivel de aplicación en el formato que internamente use el programa y es codificado de acuerdo con un protocolo estándar. Una vez que dichos que los datos han sido codificados, son pasados a la capa de transporte normalmente a través de protocolos como TCP y UDP.

- La capa de transporte: Tiene como tarea principal la de proveer comunicación punto a punto entre las aplicaciones. Los protocolos de transporte (TCP y UDP) usan el servicio de entrega de paquetes que provee la capa de Internet.

  TCP (Protocolo de control de transmisión) protocolo que proporciona un mecanismo de transporte fiable y orientado a conexión, con flujo fiable de bytes, lo que asegura que los datos llegan completos, sin daños y en orden. TCP realiza continuamente medidas sobre el estado de la red para evitar sobrecargarla con demasiado tráfico.

  UDP (User datagram protocol) es un protocolo que implementa un nivel de transporte basado en el intercambio de datagramas sin conexión que se encapsulan dentro de la parte de datos de un datagrama IP. Este servicio puede duplicar y reordenar los paquetes.

- La capa de Internet: Provee el servicio de entrega de paquetes de una máquina a otra o entre redes, por medio del protocolo de Internet (IP). La integridad de los datos no se verifica en este

nivel, por lo que el mecanismo de verificación es implementado en la capa de transporte o de aplicación.

- La capa de enlace de datos: especifica la forma como se transportan los paquetes en la capa física.
- La capa física: especifica las características físicas de la comunicación (tipo de cables, conectores, código de canales, potencia de la señal, longitud de onda, sincronización, distancias, entre otras).

Apéndice G – Funcionamiento de la Plataforma

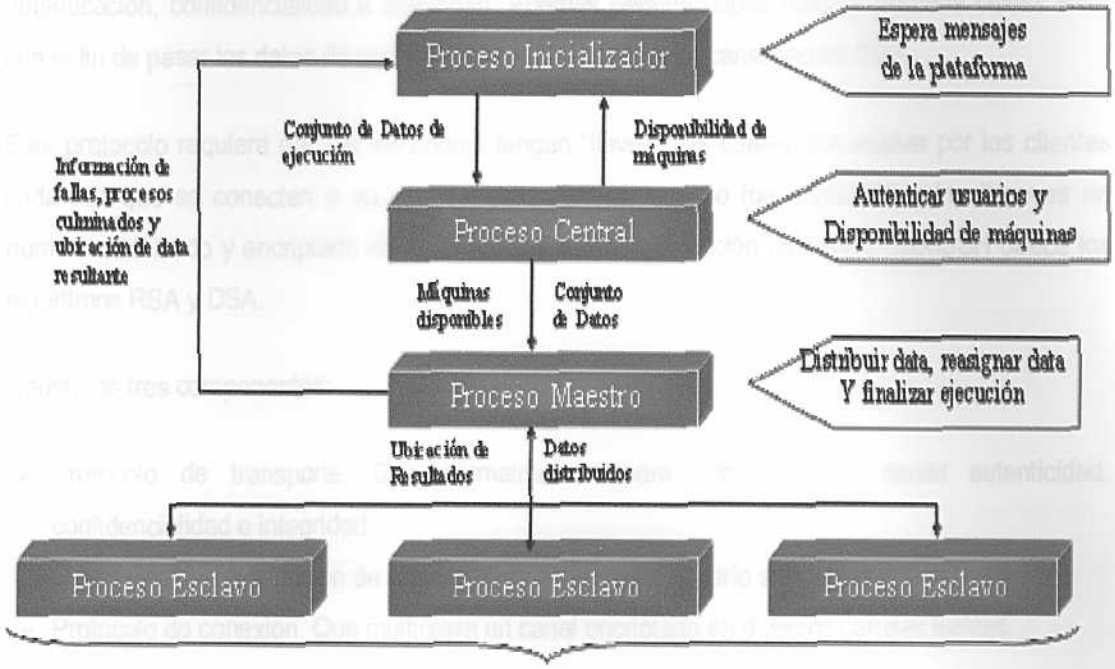

Figura G.1 Funcionamiento de la plataforma

Fuente: Elaboración propia

Apéndice H – Protocolo SSH

SSH (Secure Shell) es un protocolo para crear conexiones remotas seguras, ofreciendo autenticación, confidencialidad e integridad. Además permite copiar datos y gestiona claves RSA con el fin de pasar los datos de cualquier otra aplicación por un canal seguro SSH.

Este protocolo requiere que los servidores tengan "llaves", las cuales son usadas por los clientes cada vez que se conectan a un servidor para verificar que no fue suplantado. Una llave es un número codificado y encriptado en un archivo. Para la encriptación de llaves, OpenSSH ofrece los algoritmos RSA y DSA.

Consta de tres componentes:

- Protocolo de transporte. Que normalmente opera sobre TCP/IP dando autenticidad, confidencialidad e integridad.
- Protocolo de autenticación de usuario. Que autentica al usuario ante el servidor.
- Protocolo de conexión. Que multiplexa un canal encriptado en diversos canales lógicos.

SSH proporciona los siguientes tipos de protección:

- El cliente puede verificar que se está conectando al mismo servidor durante sesiones ulteriores.
- El cliente puede transmitir su información de autentificación al servidor en formato cifrado.
- Todos los datos enviados y recibidos durante la conexión se transfieren por medio de encriptación fuerte.

Apéndice I – Salidas de la Plataforma

```
t40006: PROCESO CENTRAL: comprobando disponibilidad de máquinas
t40006:  > Las siguientes máquinas se encuentran disponibles:
t40006: wales.CIDI2.ve
t40006: germany.CIDI.ve
t40006: scotland.CIDI.ve
t40006:  > Las siguientes maquinas se encuentran ocupadas:
t40006:  > Todas las máquinas están disponibles
```

Figura I.1 Mensajes del Proceso Central

Fuente: Elaboración propia

```
t40007: PROCESO MAESTRO: enviando data a máquinas disponibles
t40007: Los parámetros:/usr/share/pvm3/bin/LINUX/probando hola que tal>, se enviaron a: wales.CI
DI2.ve
t40007: Los parámetros:/usr/share/pvm3/bin/LINUX/probando hola que tal>, se enviaron a: germany.
CIDI.ve
t40007: Los parámetros:/root/probando1 hola que tal>, se enviaron a: scotland.CIDI.ve
t40008: PROCESO ESCLAVO N°: 262152 ha iniciado. Ejecutando: /usr/share/pvm3/bin/LINUX/probando
t80001: PROCESO ESCLAVO N°: 524289 ha iniciado. Ejecutando: /usr/share/pvm3/bin/LINUX/probando
tc0001: PROCESO ESCLAVO N°: 786433 ha iniciado. Ejecutando: /root/probando1
t40008: La ejecución del archivo </usr/share/pvm3/bin/LINUX/probando> ha finalizado.
t4000a: Los parámetros:/root/probando hola que tal>, se enviaron a: wales.CIDI2.ve
t80001: La ejecución del archivo </usr/share/pvm3/bin/LINUX/probando> ha finalizado.
t4000b: PROCESO ESCLAVO N°: 262155 ha iniciado. Ejecutando: /root/probando
t4000b: Ha ocurrido un error al intentar ejecutar el archivo: /root/probando  → Archivo no encontrado
tc0001: La ejecución del archivo </root/probando1> ha finalizado.
```

Figura I.2 Mensajes del Proceso Maestro y Procesos Esclavos

Fuente: Elaboración propia

```
Sun Jun 25 15:04:25 2006
 -> t262571: INICIALIZADOR SERVIDOR: procesando solicitud
Sun Jun 25 15:04:25 2006
 -> t262571: Archivo válido.
Sun Jun 25 15:04:25 2006
 -> t262574: PROCESO CENTRAL: comprobando disponibilidad de máquinas
Sun Jun 25 15:04:25 2006
 -> t262574:  > Las siguientes máquinas se encuentran disponibles:
Sun Jun 25 15:04:25 2006
 -> t262574: italy.gise.ve
Sun Jun 25 15:04:25 2006
 -> t262574: spain.gise.ve
Sun Jun 25 15:04:25 2006
 -> t262574:  > Las siguientes maquinas se encuentran ocupadas:
Sun Jun 25 15:04:25 2006
 -> t262574:  > Todas las máquinas están disponibles
Sun Jun 25 15:04:28 2006
 -> t262575: PROCESO MAESTRO: enviando data a máquinas disponibles
Sun Jun 25 15:04:28 2006
 -> t262575: Los parámetros:/usr/share/pvm3/bin/LINUX/probando hola que
tal>, se enviaron a: italy.gise.ve
Sun Jun 25 15:04:28 2006
 -> t262575: Los parámetros:/usr/share/pvm3/bin/LINUX/probando hola que
tal>, se enviaron a: spain.gise.ve
Sun Jun 25 15:04:31 2006
 -> t262576: PROCESO ESCLAVO Nº: 262576 ha iniciado. Ejecutando: /usr/share/
pvm3/bin/LINUX/probando
Sun Jun 25 15:04:31 2006
 -> t524289: PROCESO ESCLAVO Nº: 524289 ha iniciado. Ejecutando: /usr/share/
pvm3/bin/LINUX/probando
Sun Jun 25 15:04:37 2006
 -> t262576: La ejecución del archivo </usr/share/pvm3/bin/LINUX/probando>
ha finalizado.
```

Figura I.3 Salida del archivo logs.txt

Fuente: Elaboración propia

Apéndice J – Diagramas de Flujo de Procesos del segundo prototipo

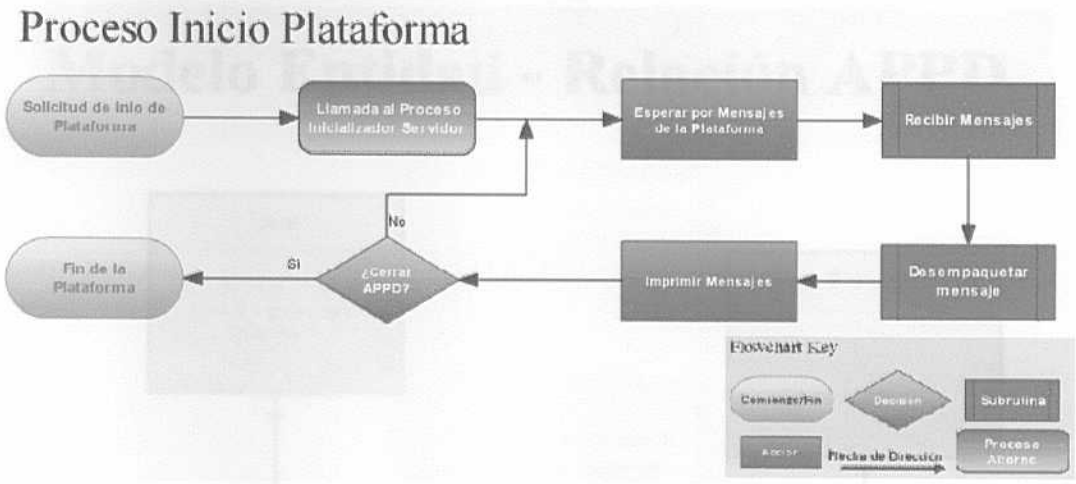

Figura J.1 Diagrama de flujo de procesos del "Proceso Inicio Plataforma"

Fuente: Elaboración propia

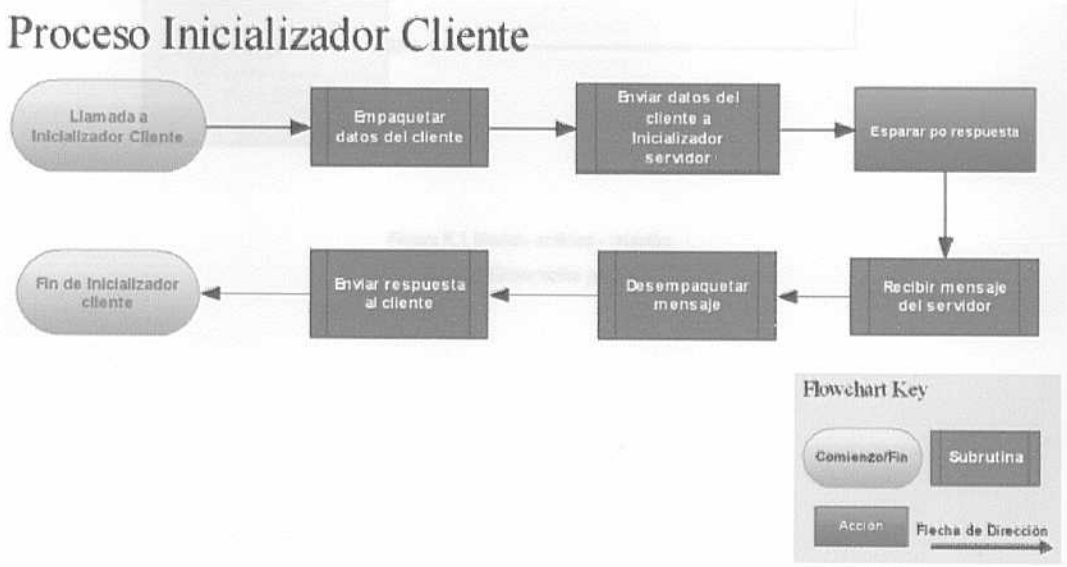

Figura J.2 Diagrama de flujo de procesos del "Proceso Inicializador Cliente"

Fuente: Elaboración propia

Apéndice K – Modelo Entidad - Relación

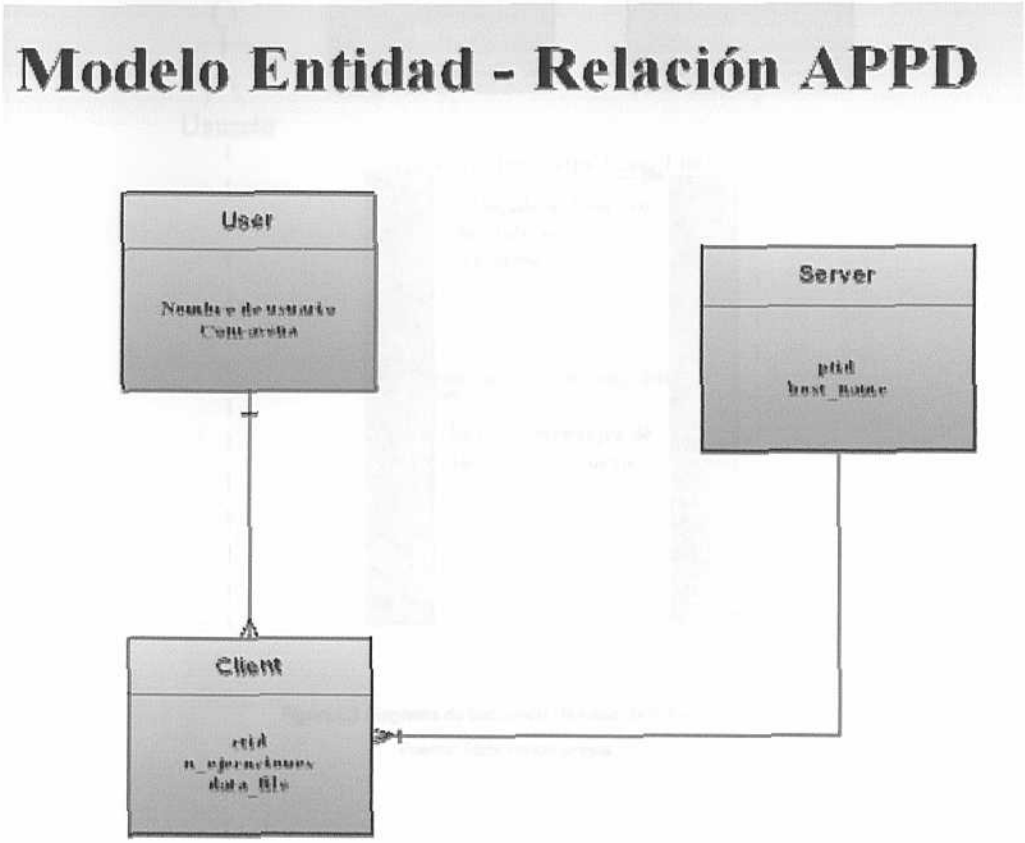

Figura K.1 Modelo entidad - relación

Fuente: Elaboración propia

Apéndice L – Diagramas de Secuencia

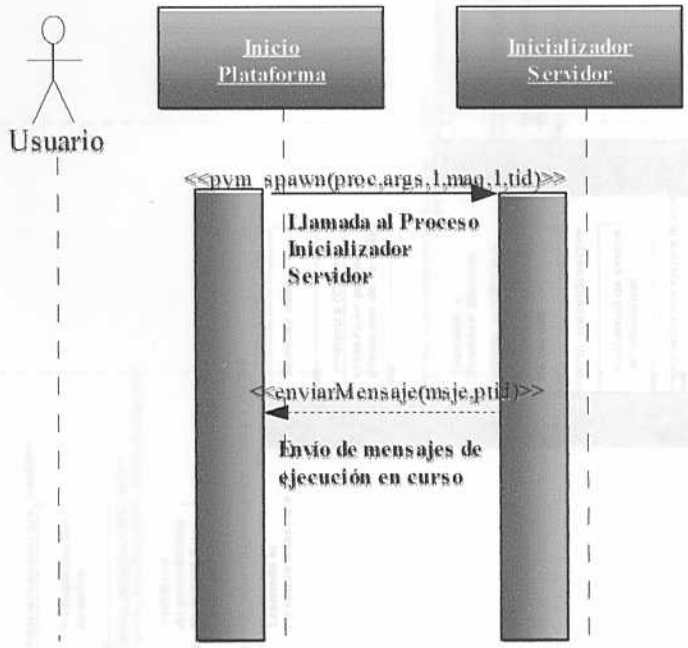

Figura L.1 Diagrama de Secuencia de Inicio de la Plataforma

Fuente: Elaboración propia

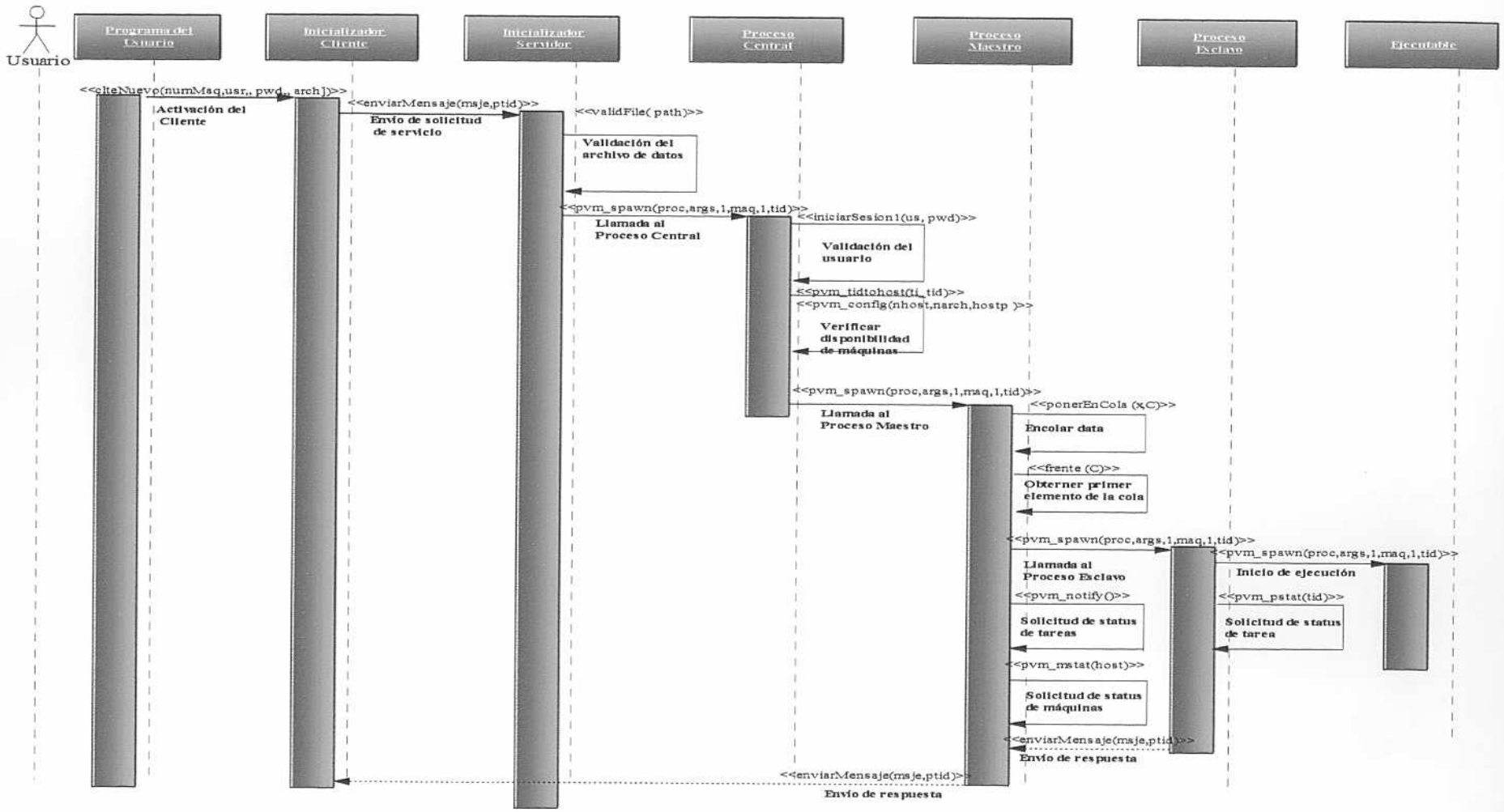

Figura L.2 Diagrama de Secuencia de Solicitud de Servicio

Fuente: Elaboración propia