UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

Desarrollo de un Ambiente de Programación Paralelo para la Ejecución de Aplicaciones en C y JAVA

Este Jurado; una vez realizado el examen del presente trabajo ha evaluado su contenido con el resultado: Veinte (20)

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

Firma:	Firma:	Firma:
Nombre: SAUL BUITRAGO	Nombre: RAUL SÁNCHEZ	Nombre: Rodolfo Campos

REALIZADO POR	Maricelys E. Blanco G.
PROFESOR GUIA	Saul Buitrago
FECHA	Septiembre, 2008

ÍNDICE GENERAL

DEDICATORIA i

AGRADECIMIENTOS ii

ÍNDICE GENERAL iii

ÍNDICE DE TABLAS x

ÍNDICE DE FIGURAS xiii

SINOPSIS xvii

Introducción 1

CAPÍTULO I. Planteamiento del Problema

I.1. Planteamiento del Problema 2

I.2. Objetivos 3

I.2.1. Objetivo General 3

I.2.2. Objetivos Específicos 3

I.3. Justificación 5

I.4. Alcances 5

I.5. Limitaciones 6

CAPÍTULO II. Marco Teórico

II.1 Sistemas Distribuidos 7

II.2 Procesamiento Paralelo 8

II.3 Cluster 8

II.4 Middleware 10

II.5 Message Passing Interface (MPI) 12

II.5.1 MPICH 13

II.5.2 mpiJava 14

II.6 Herramientas de Balanceo de Carga 14

II.6.1 Tera-scale Open-source Resource and QUEue manager (TORQUE) 14

II.6.2 Portable Batch System (OpenPBS) 15

II.7 Rendimiento 15

CAPÍTULO III. Marco Metodológico

III.1 Metodología para el Middleware: Rapid Application Development (RAD) 16

III.1.1Elementos de RAD 17

III.1.2 Aplicación de RAD 17

III.2 Guía Metodológica para el Estudio de Medición de Rendimiento 18

III.3 Guía Metodológica para el Estudio de Paralelización 19

III.3.1 Fases de la Programación Paralela 19

CAPÍTULO IV. Desarrollo

IV.1 Desarrollo del Middleware: Primer Prototipo 20

IV.1.1 Análisis y Planificación de Requerimientos 20

IV.1.2 Diseño 23

IV.1.2.1 Diseño de la Comunicación entre los Componentes del Middleware 23

IV.1.2.2 Diseño de Paradigma de Programación Paralela 24

IV.1.2.3 Diseño de Archivo de Datos 24

IV.1.2.4 Diseño de la Aplicación 26

IV.1.3 Construcción 28

IV.1.3.1 Construcción de Procesos 28

IV.1.4 Pruebas 31

IV.1.4.1 Pruebas del Middleware Primer Prototipo 31

IV.1.5 Post- Actividades del Proyecto 35

IV.2 Desarrollo del Middleware: Segundo Prototipo 36

IV.2.1 Planificación y Análisis de Requerimientos 37

IV.2.1.1 Hardware 37

IV.2.1.2 Ambiente de Trabajo 38

IV.2.1.3 Arquitectura del Cluster 38

IV.2.1.4 Arquitectura de la Aplicación 39

IV.2.2 Diseño 39

IV.2.2.1 Diseño de archivos de datos 39

IV.2.2.2 Diseño de la Aplicación 40

IV.2.3 Construcción 41

IV.2.3.1 Construcción de Componentes 41

IV.2.3.1.1 Acceso 41

IV.2.3.1.2 Clases y archivos de instrucciones 42

IV.2.3.1.3 Interfaz 45

IV.2.4 Pruebas 47

IV.2.4.1 Verificación de NFS 48

IV.2.4.2 Verificación de PVM 49

IV.2.4.3 Pruebas de control de fallos 50

IV.2.4.4 Pruebas de Interfaz 51

IV.2.5 Post- Actividades del Proyecto 51

IV.3 Desarrollo de Estudios de Rendimiento 52

IV.3.1 Objetivos y Métricas 52

IV.3.2 Parámetros y Factores de las Mediciones 53

IV.3.3 Instalaciones y Configuraciones 55

IV.4 Desarrollo de Estudio de Paralelización de DECA 56

IV.4.1 Planteamiento 56

IV.4.2 Esquema General de la Aplicación 57

IV.4.3 Análisis 58

CAPÍTULO V. Resultados 62

CAPÍTULO VI. Conclusiones y Recomendaciones 65

BIBLIOGRAFÍA 70

GLOSARIO 75

APÉNDICES 81

Apéndice A- Sistemas Distribuidos

A.1 Combinaciones Comunes de Software y Hardware 81

A.2 Tipos de Transparencia 82

Apéndice B – Procesamiento Paralelo

B.1 Clasificación 83

B.2 Características 83

Apéndice C - Clusters

C1. Características 85

C.2 Tipos de Software 85

C.3 Servicios 87

C.4 Middleware 87

Apéndice D – Paradigma de Paso de Mensajes MPI (Message Passing Interface)

D.1 Parallel Virtual Machine (PVM) 88

D.2 Comparaciones entre PVM y MPI 88

D.3 Origen de MPI 91

D.4 Implementación de MPICH 92

D.5 mpiJava 92

Apéndice E – Herramientas de Balanceo de Carga

E.1 TORQUE 93

E.2 OpenPBS 94

E.3 Comparaciones entre TORQUE y OpenPBS 95

Apéndice F – Rendimiento

F.1 Métricas Generales para Aplicaciones Paralelas 97

F.2 Monitores y Rastros de Ejecución 98

F.3 Necesidad del Control y de la Evaluación del Rendimiento 99

F.4 Mediciones del Rendimiento 101

Apéndice G - Planificación y Análisis de Requerimientos

G.1 Hardware 103

G.2 Ambiente de Trabajo 103

G.3 Arquitectura del Cluster 106

G.4 Arquitectura de la Aplicación 106

G.5 Arquitectura de Seguridad 109

G.6 Tolerancia a Fallos 109

G.7 Cuadro Descriptivo de Casos de Uso 110

Apéndice H - Instalaciones y Configuraciones

H.1 Instalación y Configuración de MPICH2 117

H.2 Instalación y Configuración de Java 118

H.3 Instalación y Configuración de mpiJava 119

H.4 Instalación y Configuración de TORQUE 120

H.5 Instalación de apache, mySQL y PHP en Gibraltar 121

Apéndice I – Pruebas y Verificaciones

I.1 Pruebas de MPICH 121

I.2 Pruebas de Java 124

I.3 Pruebas de mpiJava 124

I.4 Pruebas de TORQUE 125

I.5 Verificación de SSH y TCP/IP 126

Apéndice J – Diagrama de clases 128

Apéndice K – Pruebas del Segundo Prototipo 131

Apéndice L – Manual de Usuarios 133

Apéndice M – Mediciones y Resultados

M.1 EX_IN 134

M.1.1 Mediciones 134

M.1.2 Resultados 137

M.1.3 Gráficas 140

M.2 DECA 141

M.2.1 Mediciones 141

M.2.2 Resultados 143

M.2.3 Gráficas 145

Apéndice N - JAVA y Base de Datos 146

Apéndice O – Archivos del Sistema 150

Apéndice P – DECA y EX_IN 153

ÍNDICE DE TABLAS

Tabla 1. Elementos de RAD 17

Tabla 2. Fases del modelo de desarrollo de RAD 17

Tabla 3. Fases para los estudios de mediciones de rendimiento 18

Tabla 4. Secciones Consideradas para el estudio de paralelización 19

Tabla 5. Relación de usuarios con el sistema 22

Tabla 6. Tipos de transparencia en un sistema distribuido 82

Tabla 7. Características de PVM y MPI 89

Tabla 8. Ventajas de PVM y MPI 90

Tabla 9. Desventajas de PVM y MPI 90

Tabla 10. Componentes de TORQUE y OpenPBS 95

Tabla 11. Ventajas de TORQUE y OpenPBS 96

Tabla 12. Desventajas de TORQUE y OpenPBS 97

Tabla 13. Cuadro Descriptivo CU 1 110

Tabla 14. Cuadro Descriptivo CU 2 111

Tabla 15. Cuadro Descriptivo CU 3 112

Tabla 16. Cuadro Descriptivo CU 4 113

Tabla 17. Cuadro Descriptivo CU 5 114

Tabla 18. Cuadro Descriptivo CU 6 115

Tabla 19. Cuadro Descriptivo CU 7 116

Tabla 20. Medidas globales de EX_IN, secuencial y simulado paralelo, sin y con bucle for 134

Tabla 21. Medidas de EX_IN sin bucle, en grupos secuencial y simulado paralelo 135

Tabla 22. Medidas de EX_IN con bucle, en grupos secuencial y simulado paralelo 136

Tabla 23. Promedios de EX_IN para ANOVA, experimento con 2 factores 137

Tabla 24. SST de EX_IN, experimento con 2 factores 137

Tabla 25. Resumen de promedios EX_IN en ANOVA 138

Tabla 26. Análisis de varianza de EX_IN 138

Tabla 27. Resumen de ANOVA para la data sin bucle de EX_IN 138

Tabla 28. Análisis de varianza para la data sin bucle de EX_IN 139

Tabla 29. Resumen de ANOVA para la data con bucle de EX_IN 139

Tabla 30. Análisis de varianza para la data con bucle de EX_IN 139

Tabla 31. Medidas de control de rendimiento DECA 142

Tabla 32. Análisis de varianza de DECA 143

Tabla 33. Determinación de las y, q0, qa, qb, qc, qab, qac, qbc, qabc 143

Tabla 34. Determinación de las fórmulas y*q0, y*qa, y*qb, y*qc, y*qab, y*qac, y*qbc, y*qabc 143

Tabla 35. Determinación de las fórmulas y*q0/8, y*qa/8, y*qb/8, y*qc/8, y*qab/8, y*qac/8, y*qbc/8, y*qabc/8 144

Tabla 36. Determinación de las fórmulas (y*q0/8) ^2, (y*qa/8) ^2, (y*qb/8) ^2, (y*qc/8) ^2, (y*qab/8) ^2, (y*qac/8) ^2, (y*qbc/8) ^2, (y*qabc/8) ^2 144

Tabla 37. SSA, SSB, SSC, SSAB, SSAC, SSBC, SSABC, SST 144

Tabla 38. Porcentajes, SSA, SSB, SSC, SSAB, SSAC, SSBC, SSABC, SST 144

ÍNDICE DE FIGURAS

Figura 1. Modelo General del Proyecto PISACOTA 3

Figura 2. Middleware enmarcado en un cluster 10

Figura 3. Ubicación del middleware de cluster 11

Figura 4. Ubicación de MPI en el proceso de programación de aplicaciones paralelas 13

Figura 5. Capas de Software 14

Figura 6. Esquema de RAD 18

Figura 7. Diagrama de Casos de Uso de la plataforma 22

Figura 8. Estructura del archivo "data" 25

Figura 9. Diagrama de Secuencia 27

Figura 10. Respuesta de la prueba de funcionamiento 32

Figura 11. Respuesta ante la caída de un nodo 34

Figura 12. Directorio de tareas batch ante la caída de un nodo 35

Figura 13. Nueva Arquitectura del Cluster 39

Figura 14. Estructura del archivo "salida" 40

Figura 15. Interfaz Web Informativa 46

Figura 16. Ejemplo de Reinicio del Sistema 48

Figura 17. Ejemplo de uso de aplicación de PVM con TORQUE, archivo "problem.sh" y "problem.sh.o230" 49

Figura 18. Prueba del módulo de TORQUE 50

Figura 19. Ventana de interfaz gráfica 51

Figura 20. Código Secuencial del método Procesar en DemoUpBack 61

Figura 21. Clusters. Cluster a nivel de sistema y nivel de aplicación 86

Figura 22. Herencia de clases 92

Figura 23. Arquitectura del cluster 106

Figura 24. Modelo de Dominio de la plataforma 107

Figura 25. Arquitectura de la aplicación 108

Figura 26. Instalación y configuración de MPICH2 118

Figura 27. Instalación y configuración de Java 118

Figura 28. Instalación y configuración de mpiJava 119

Figura 29. Instalación y configuración de TORQUE 120

Figura 30. Comandos de instalación de apache, mysql y php con yum en Gibraltar 121

Figura 31. Primera prueba MPICH 122

Figura 32. Segunda prueba MPICH 122

Figura 33. Tercera prueba MPICH 122

Figura 34. Cuarta prueba MPICH 123

Figura 35. Quinta prueba MPICH 123

Figura 36. Prueba de Java 124

Figura 37. Primera prueba de mpiJava 124

Figura 38. Segunda prueba de mpiJava 124

Figura 39. Prueba de TORQUE 125

Figura 40. Archivo de Resultado de la prueba de TORQUE: vamos.sh.o2 125

Figura 41. Verificación de SSH 126

Figura 42. Verificación de TCP/IP 127

Figura 43. Diagrama de Clases 128

Figura 44. Diagrama de Clases- Interfaz 129

Figura 45. Diagrama de Clases- Salida- Torque 130

Figura 46. Archivo errorPBSGermany.txt 131

Figura 47. Directorio de tareas batch después de la caída y recuperación de un nodo 131

Figura 48. Fragmento del archivo "final", resultado de la ejecución del middleware en Gibraltar 132

Figura 49. Fragmento del archivo "manual_usuario" 133

Figura 50. Valores promedios de EX_IN con bucle 140

Figura 51. Valores promedios de EX_IN sin bucle 140

Figura 52. Valores promedios de DECA 145

Figura 53. Modelo de Entidad- Relación de APPD_bd 146

Figura 54. Archivo .bashrc 148

Figura 55. Instalación y selección de la versión del JDK a utilizar en Germany 149

Figura 56. Archivo /etc/hosts 150

Figura 57. Archivo mpd.hosts 150

Figura 58. Archivo mpd.conf 150

Figura 59. Archivo hosts.allow en wales 151

Figura 60. Archivo hosts.deny en wales 151

Figura 61. Archivo config de TORQUE en Germany 152

Figura 62. Archivo nodes de TORQUE 152

Figura 63. Archivo server_name de TORQUE 152

Figura 64. Archivo habilitado para usuarios en apache 152

Figura 65. Aplicación DECA resultados 153

Figura 66. Aplicación DECA configuración 153

Figura 67. a: Imagen a. DECA 154

Figura 68. b: Imagen b. DECA 155

Figura 69. A: Fondo 1. DECA 156

Figura 70. B: Fondo 2. DECA 157

Figura 71. Aplicación EX_IN resultados 158

SINOPSIS

En este trabajo Especial de Grado se diseña y desarrolla una aplicación que funciona como middleware en un cluster de cómputo permitiendo alcanzar un mayor nivel de transparencia en la ejecución de aplicaciones implementadas en C y Java.

Adicionalmente se realiza un control de rendimiento en un nodo único del Sistema de Detección de Tipo y Cantidad de Vehículos Presentes en una Imagen Digital (DECA), y se establece una comparación de rendimiento del programa EX_IN, el cual calcula los mínimos de una función en un dominio determinado, ejecutado de forma secuencial y luego de forma simulada paralela en un nodo único con la finalidad de sugerir formas de mejorar el rendimiento actual para cada caso.

Como resultado del desarrollo se obtuvo un middleware de cluster de cómputo, capaz de permitir la ejecución de aplicaciones tanto paralelas como secuenciales en C y Java, que utilicen las librerías de paso de mensajes Parallel Virtual Machine (PVM) o Message Passing Interface (MPI).

Adicionalmente la ejecución de ciertos comandos de sistema también es factible de procesar por la aplicación resultante, contando con una interfaz de usuario, una base de datos para el control de acceso, el uso de MPI para

la distribución de las tareas entre los nodos y utilizando una herramienta llamada Tera-scale Open-source Resource and QUEue manager (TORQUE) para el procesamiento batch de las tareas a ejecutar tomando en cuenta el balance de carga del cluster.

También, con el análisis de las mediciones de los estudios de las aplicaciones antes mencionadas se llegó a obtener ideas claras sobre el comportamiento de las mismas en el escenario considerado así como propuestas para mejorar su rendimiento haciendo uso del procesamiento paralelo.

INTRODUCCIÓN

Los clusters se han definido como "grupos de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio". Una de las utilidades más consideradas radica precisamente en el poder de cómputo que se puede obtener en un tiempo menor a comparación a la utilización de una sola máquina. Desde aplicaciones de súper cómputo y software de misiones críticas, servidores Web y comercio electrónico, hasta bases de datos de alto rendimiento, son parte de la lista de usos de los clusters.

Para lograr que ante el usuario un aglomerado de máquinas parezca una sola se debe contar con una o múltiples plataformas que logren gestionar el uso del mismo para las tareas a ejecutar.

El presente trabajo describe el proceso de creación de un middleware para un cluster de cómputo que permita obtener un alto grado de transparencia para el usuario en la ejecución de implementaciones en C y Java. Adicionalmente se presenta un estudio sobre el comportamiento de un par de aplicaciones implementadas en dichos lenguajes en relación a su rendimiento actual y con la posibilidad de ser implementadas en un cluster de cómputo.

CAPÍTULO I
Planteamiento del Problema

I.1 Planteamiento del Problema

El Centro de Investigación y Desarrollo de Ingeniería (CIDI) responde a la inquietud de la Facultad de Ingeniería de impulsar las inquietudes de investigación de muchos profesores (www.universia.edu.ve). De una de estas inquietudes nace un proyecto, el cual se encuentra en constante crecimiento gracias al trabajo de docentes y estudiantes comprometidos con el CIDI y con nuestra casa de estudio, la Universidad Católica Andrés Bello (UCAB). Dicho proyecto se denomina Prototipo de Plataforma Telemática para Simulación de Tráfico Vehicular (PISACOTA), en vías expresas de la ciudad de Caracas.

La arquitectura de la plataforma cuenta con un cluster de cómputo encargado de manejar las aplicaciones necesarias para garantizar su funcionamiento. Hasta los momentos, el cluster ya mencionado se limita a la ejecución de códigos pre- compilados en lenguaje C y Fortran, sin embargo esta característica es una limitante al tratarse con aplicaciones desarrolladas en JAVA que también requieren ser ejecutadas en dicha estructura física.

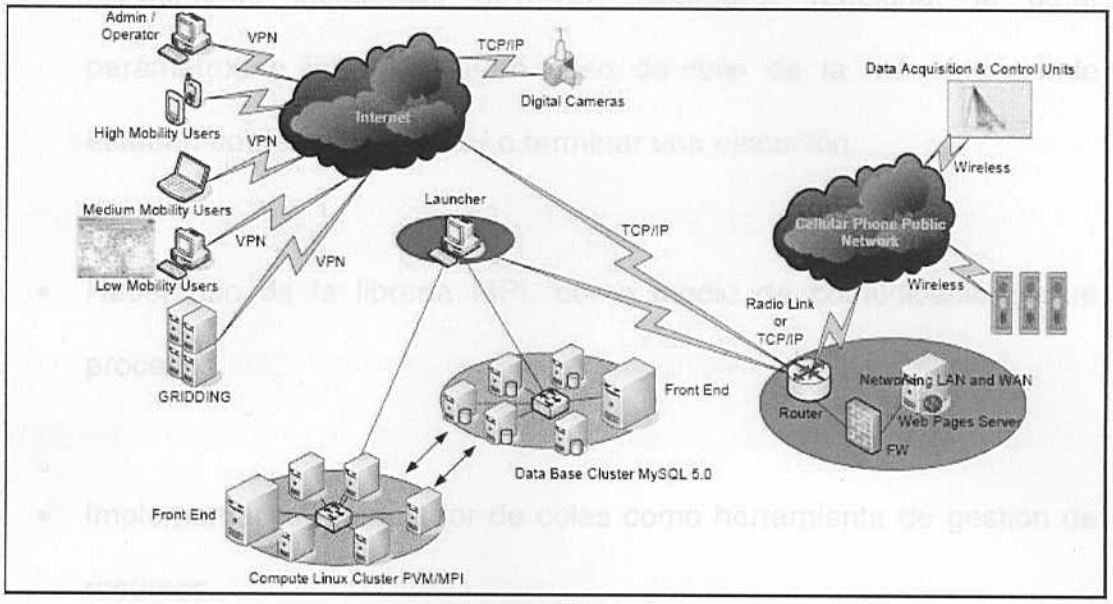

Figura 1. Modelo general del proyecto PISACOTA. *Fuente*: Trabajo de Grado "Unidad de Transmisión Inalámbrica para Estaciones Automatizadas de Adquisición de Datos deTráfico Automotor".

I.2 Objetivos

I.2.1 Objetivo General

Desarrollar un ambiente de programación paralelo sobre un cluster de cómputo que sirva como soporte a aplicaciones en lenguaje C y JAVA.

I.2.2 Objetivos Específicos

- Diseñar, desarrollar e implantar el ambiente de procesamiento paralelo, 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.

- Hacer uso de la librería MPI, como medio de comunicación entre procesos.
- Implementar un manejador de colas como herramienta de gestión de recursos.
- Realizar pruebas iniciales de rendimiento en un nodo único de aplicaciones que forman el proyecto PISACOTA implementadas en C y JAVA respectivamente.
- Realizar un estudio preliminar del acondicionamiento de la aplicación DECA para que pueda ser ejecutada en el ambiente paralelo a desarrollar.
- Realizar pruebas sobre del funcionamiento del cluster con ejemplos sencillos.

I.3 Justificación

La idea de la creación del ambiente propuesto surge como un requisito para el cumplimiento de requerimientos, traducidos a sistemas, del proyecto PISACOTA, en donde se propone la conformación de estaciones encargadas de la adquisición de datos del tráfico terrestre para luego ser procesados y lograr así la optimización de la configuración de las vías de tránsito terrestre mediante los análisis que se realicen de dichos resultados. Para que los sistemas desarrollados se ejecuten en un ambiente paralelo, aprovechando así el mayor poder de cómputo disponible, deben ser acondicionados y el cluster que lo soporte debe ser configurado para aceptar estas aplicaciones cuando estén listas para ello.

I.4 Alcances

La aplicación de procesamiento paralelo a desarrollar contará con la librería de comunicación MPI y la herramienta de gestión de recursos seleccionada para la ejecución de aplicaciones desarrolladas en lenguaje C y Java.

I.5 Limitaciones

- 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 salidas, serán los resultados de la ejecución de los procedimientos a través de vectores o archivos de textos.

CAPÍTULO II
Marco Teórico

A continuación se presentan algunos conceptos necesarios para la compresión de este documento.

II.1 Sistemas Distribuidos

Gracias a los considerables avances tanto en redes de área local (LAN) como amplia (WAN), las cuales permiten conectar respectivamente cientos de computadoras en un mismo edificio o millones de máquinas alrededor del mundo en períodos de tiempo muy cortos, han hecho posible reunir sistemas de cómputo compuestos por un gran número de CPU mediante una red de alta velocidad.

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

Se pueden distinguir recursos de hardware y software en los sistemas distribuidos, los cuales están distribuidos físicamente y funcionan gracias a una red de comunicaciones. Un aspecto clave en el diseño de este tipo de sistemas es la transparencia, es decir, la acción de ocultar la naturaleza distribuida del sistema y presentarla como si se tratara de un solo equipo.

La tendencia actual a la globalización exige que los sistemas y aplicaciones informáticas vayan más allá de lo alcanzable por cualquier ordenador aislado, por muy potente que éste sea. Ello hace que los sistemas distribuidos se vayan convirtiendo en el modelo generalizado. En el Apéndice A se explica un poco más sobre los mismos.

II.2 Procesamiento Paralelo

Varias máquinas conectadas a través de una red forman sistemas de computación conocidos como procesadores paralelos. Éstos cuentan con el software necesario para que las unidades procesen la data de manera conjunta comunicándose entre sí gracias al uso de memoria compartida o con métodos de paso de mensajes.

La taxonomía de Flynn (1972) resulta ser uno de los esquemas de clasificación para los sistemas de cómputo con varios CPU más citado. En ella resaltan dos características: el número de flujo de instrucciones y el número de flujo de datos. Consultar Apéndice B para más detalles.

II.3 Cluster

Los clusters se definen como un grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio.

Juegan hoy en día un papel importante en la solución de problemas de las ciencias, las ingenierías y del comercio moderno.

La tecnología de clusters ha evolucionado en apoyo de actividades que van desde aplicaciones de supercómputo y software de misiones críticas, servidores Web y comercio electrónico, hasta bases de datos de alto rendimiento, entre otros usos.

La construcción de los ordenadores del cluster es más fácil y económica debido a su flexibilidad: pueden tener todos la misma configuración de hardware y sistema operativo (cluster homogéneo), diferente rendimiento pero con arquitecturas y sistemas operativos similares (cluster semi-homogéneo), o tener diferente hardware y sistema operativo (cluster heterogéneo).

Para que un cluster funcione como tal, no basta sólo con conectar entre sí los ordenadores, sino que es necesario proveer un sistema de manejo del cluster, el cual se encargue de interactuar con el usuario y los procesos que corren en él para optimizar el funcionamiento.

El cómputo con clusters surge como resultado de la convergencia de varias tendencias actuales que incluyen la disponibilidad de microprocesadores económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas de software para cómputo distribuido de alto rendimiento, así como la creciente necesidad de potencia

computacional para aplicaciones que la requieran. Para mayor información de los clusters consultar Apéndice C.

II.4 Middleware

El middleware es la diferencia clave entre una red de PCs y un cluster. Es el nivel de software que ejecuta por encima del sistema operativo y que proporciona al usuario una imagen de sistema único.

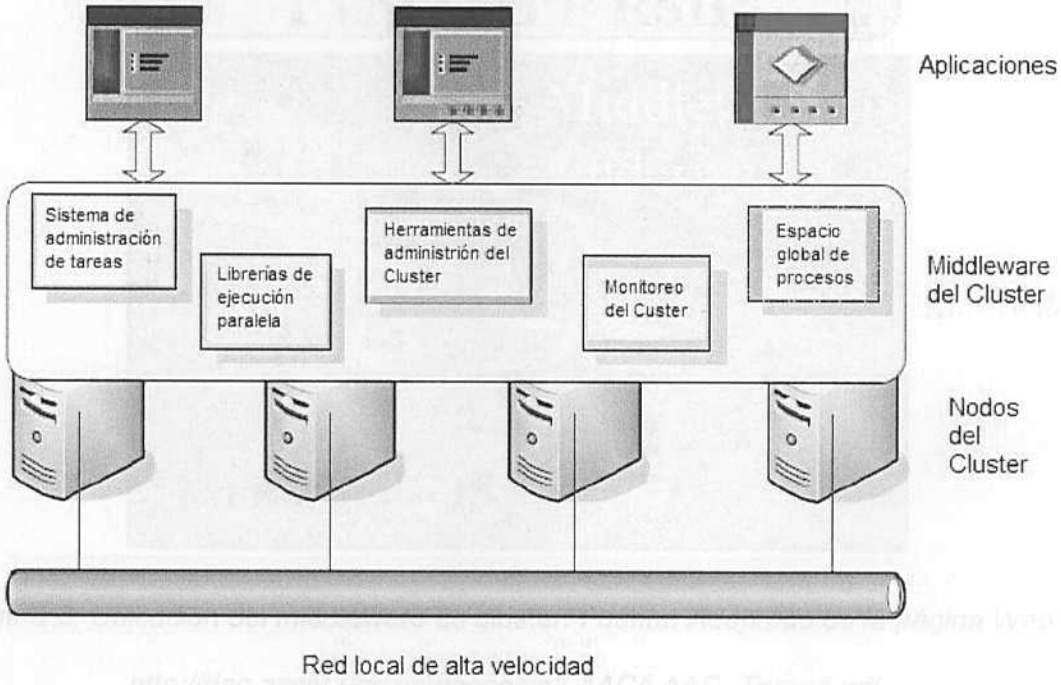

Figura 2. Middleware enmarcado en un cluster. *Fuente:* Adaptado de la página Web http://dac.escet.urjc.es/docencia/LAAC/LAAC_Tema4.pdf.

Permite un acceso uniforme a los distintos nodos de cómputo independientemente del sistema operativo que implemente cada uno de ellos. Gestiona los fallos de los nodos individuales y el equilibrio de carga de trabajo.

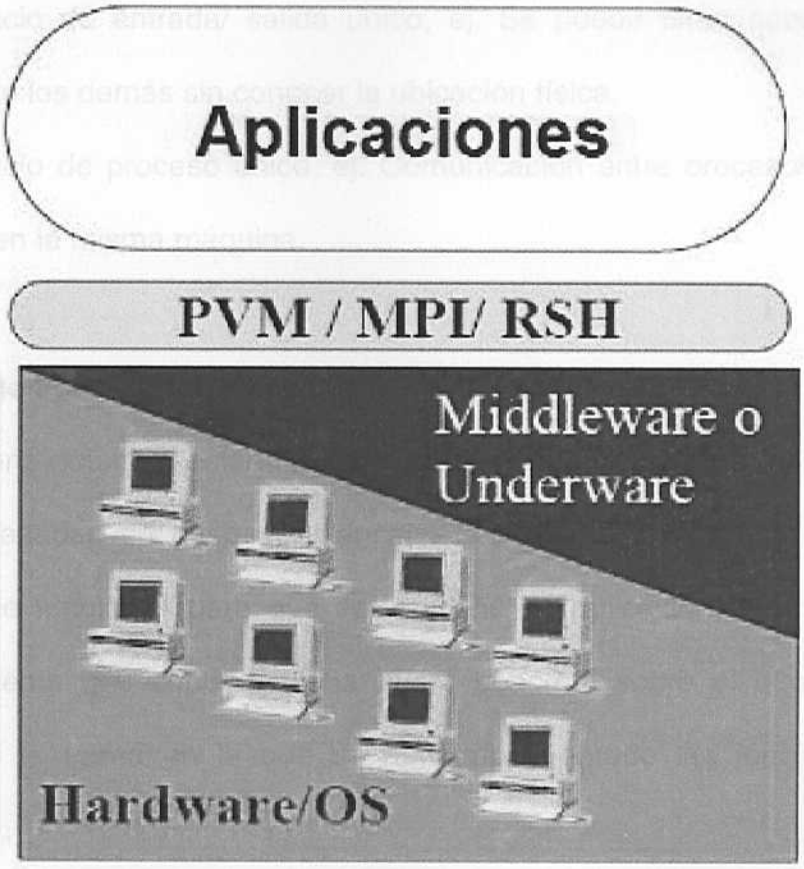

Figura 3. Ubicación del middleware de cluster. *Fuente: Adaptado de la página Web http://dac.escet.urjc.es/docencia/LAAC/LAAC_Tema4.pdf.*

El middleware contiene dos subsistemas, el de *Imagen Única,* el cual trata de la apariencia de un sistema único frente a usuarios, aplicaciones y la red, y el de *Disponibilidad,* que hace referencia a la probabilidad de que un

servicio funcione adecuadamente en cualquier momento. Entre algunos de los objetivos que debe cubrir el middleware idealmente se encuentran los siguientes:

Jerarquía de ficheros única, ej. NFS.

Interfaz de usuario único, ej. Entorno único.

Espacio de entrada/ salida único, ej. Se puede tener acceso a los nodos desde los demás sin conocer la ubicación física.

Espacio de proceso único, ej. Comunicación entre procesos como si estuvieran en la misma máquina.

II.5 Message Passing Interface (MPI)

Es una interfaz estandarizada para la realización de aplicaciones paralelas basadas en paso de mensajes. Le otorga al programador una colección de funciones para que éste diseñe su aplicación, sin que tenga necesariamente que conocer el hardware concreto sobre el que se va a ejecutar, ni la forma en la que se han implementado las funciones que emplea.

El objetivo principal de MPI es lograr obtener un grado de portabilidad comparable al de un lenguaje de programación que permita ejecutar de manera transparente, aplicaciones sobre sistemas principalmente heterogéneos.

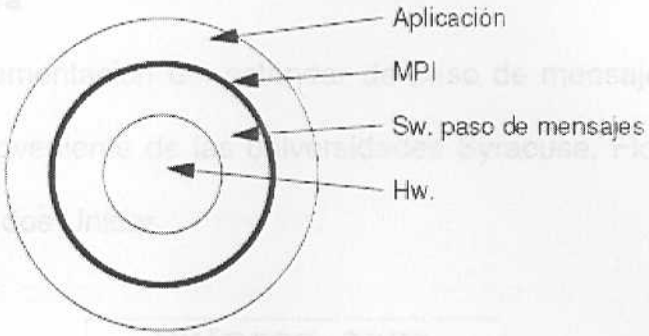

Figura 4. Ubicación de MPI en el proceso de programación de aplicaciones paralelas. *Fuente:* Disponible en http://www.sc.ehu.es/acwmialj/edumat/mpi.pdf.

II.5.1 MPICH

MPICH es una de las implementaciones más robustas de MPI, habiendo evolucionado junto con el estándar, y estando disponible para una gran cantidad de plataformas. MPICH ha seguido el desarrollo de la especificación MPI y actualmente está disponible de manera portable para una gran cantidad de plataformas, entre las que se incluyen sistemas Unix comerciales (Solaris, HP UX, AIX, IRIX), máquinas masivamente paralelas (Intel Paragon, Cray) y variantes libres de Unix (Linux, BSD). También soporta arquitecturas SMP, MPP, redes de estaciones y *clusters*.

II.5.2 mpiJava

Es una implementación del estándar de paso de mensajes MPI para Java. Desarrollo proveniente de las universidades Syracuse, Florida State e Indiana, en los Estados Unidos.

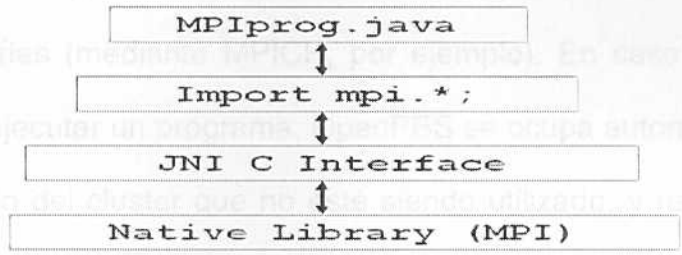

Figura 5. Capas de Software. *Fuente*: Disponible en http://ipdps.cc.gatech.edu/1999/java/baker.pdf.

Consultar Apéndice D.5 para apreciar la herencia de clases.

II.6 Herramientas de Balanceo de Carga

Son gestores de recursos que proporcionan un mecanismo para encolar, controlar y ejecutar trabajos en un recurso compartido proporcionando un acceso centralizado a una serie de recursos distribuidos, de una manera inteligente, segura, justa y eficaz.

II.6.1 Tera-scale Open-source Resource and QUEue manager (TORQUE)

Torque es un gestor de recursos que proporciona control sobre trabajos batch y nodos de cómputo distribuidos. Utiliza un mecanismo de

colas para ejecución de trabajos que respeta los criterios de prioridad configurados.

II.6.2 Portable Batch System (OpenPBS)

OpenPBS es capaz de ejecutar programas que corran en una sola máquina o en varias (mediante MPICH, por ejemplo). En caso de que un usuario necesite ejecutar un programa, OpenPBS se ocupa automáticamente de buscar un nodo del cluster que no esté siendo utilizado, y realizar dicha ejecución en el nodo. De igual manera, en caso de querer ejecutar una aplicación distribuida, buscará el número necesario de nodos para realizar la ejecución. En caso de que actualmente no pueda atenderse la ejecución de un programa por estar todos los nodos ocupados, OpenPBS introduce la petición en una cola en espera de que se liberen los recursos necesarios. Consultar Apéndice E para mayor información.

II.7 Rendimiento

Es la recolección y análisis de información relativa al rendimiento del sistema existente. Permite localizar embotellamientos con rapidez. Puede ayudar a decidir la forma de mejorar el rendimiento. El procesamiento paralelo- distribuido constituye una tendencia importante que afecta los factores de rendimiento del software. Para obtener mayores detalles ver Apéndice F.

CAPÍTULO III
Marco Metodológico

III.1 Metodología para el Middleware: Rapid Application Development (RAD)

Es un proceso de desarrollo de software, en inglés: software development process, desarrollado inicialmente por James Martin en 1980. El método comprende el desarrollo interactivo, la construcción de prototipos y el uso de utilidades conocidas como Computer Aided Software Engineering (CASE).

Tradicionalmente, el desarrollo rápido de aplicaciones tiende a englobar también la usabilidad, utilidad y la rapidez de ejecución.

Su selección se determinó por la naturaleza del trabajo a realizar, ya que tanto los elementos como fases constitutivas de RAD la hacen ideal para la ejecución del mismo, principalmente, permitiendo monitoreo constante del progreso con el uso de herramientas CASE, flexibilidad de cambios y mayor compresión de requerimientos con la elaboración de prototipos y tomas de decisiones rápidas y más efectivas debido a una relación más cercana y constante con el cliente.

III.1.1 Elementos de RAD

Elemento	Descripción
Prototipado	Constituye un aspecto importante para el diseño y ajustes de requerimientos. La idea es construir versiones completas con alcances limitados en períodos de tiempo cortos. Las herramientas CASE constituyen un insumo relevante para este fin.
Desarrollo Iterativo	Consiste en la realización de diversas entregas, versiones funcionales y continuas del sistema. Con cada una de ellas el cliente puede apreciar los avances, ajustar requerimientos y aclarar dudas.
Time Boxing	Es el tiempo de estimación de las entregas a realizar hasta alcanzar la última versión. Éste debe ser lo más corto y preciso posible.
Miembros del Grupo	Se recomienda un grupo reducido, con experiencia, motivación y capaz de cumplir diversos roles. La presencia del cliente también es muy importante sobre todo al inicio del proyecto para la recopilación de requerimientos.
Acercamiento de la Gerencia	Es un elemento decisivo en la reducción de riesgos, definición de expectativas y para reducir malentendidos y descuidos.
Herramientas	RAD motiva el uso de la última tecnología como apoyo para la ejecución de los proyectos y lograr así un desarrollo rápido y sostenido.

Tabla 1. Elementos de RAD. *Fuente*: Elaboración propia.

III.1.2 Aplicación de RAD: Fases

Fases	Descripción
Análisis y Planificación de Requerimientos	Se establecen las solicitudes o requerimientos y se definen las interacciones entre procesos y data mediante acuerdos con el cliente.
Diseño	Se detallan más los requerimientos mediante el encuentro de usuarios finales, se planifica, se diseña nuevos componentes y se identifican los reusables.
Construcción	Desarrollo, pruebas y redefinición de requerimientos. Construcción de prototipos completos pero cortos. Reuniones con el clientes para muestra de avances.
Implementación	Fase de pruebas, entregas finales. Elaboración de manuales.
Post- Actividades del Proyecto	Consiste en aprender de lo ejecutado para aplicar buenas prácticas y corregir las malas en proyectos futuros.

Tabla 2. Fases del modelo de desarrollo de RAD. *Fuente:* Elaboración propia

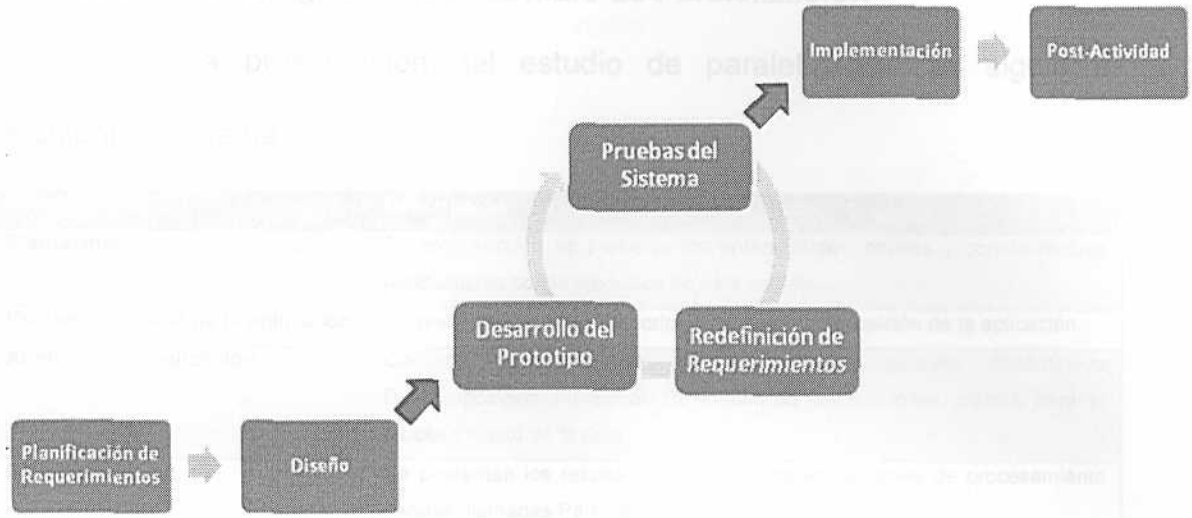

Figura 6. Esquema de RAD. *Fuente*: Elaboración propia.

III.2 Guía Metodológica para el Estudio de Medición de Rendimiento

Para la presentación de los estudios de rendimiento se siguieron las siguientes fases:

Fases	Descripción
Definición de Objetivos y Métricas	Definición de objetivos principal y específicos para darle sentido a la medición, se establecen los criterios para comparar las prestaciones.
Definición de Parámetros y Factores	Se determinan las características del sistema, y la carga de trabajo a la cual está sometido.
Análisis	Se sintetizan los datos de las medidas, se extraen conclusiones.
Resultados	Se muestran los resultados. Pueda que se repita el ciclo desde este punto.

Tabla 3. Fases para los estudios de mediciones de rendimiento.

Fuente: Elaboración propia

III.3 Guía Metodológica para el Estudio de Paralelización

Para la presentación del estudio de paralelización se siguió el siguiente esquema:

Sección	Descripción
Planteamiento	En esta sección se presenta los antecedentes, causas y consecuencias relacionadas con la ejecución de este estudio.
Esquema general de la aplicación	Se presenta una breve descripción sobre la composición de la aplicación.
Análisis de la aplicación	Coincide con la fase de programación paralela denominada Descomposición Funcional. Se identifican las funciones críticas para el procesamiento de la data.
Resultados	Se presentan los resultados, enmarcados en las fases de procesamiento paralelo llamadas Partición y Localización.

Tabla 4. Secciones Consideradas para el estudio de paralelización. *Fuente:* Elaboración propia

III.3.1 Fases de la Programación Paralela

Descomposición Funcional: Identificar las funciones que debe realizar la aplicación y las relaciones entre ellas.

Partición: Distribución de las funciones en procesos y esquema de comunicación entre procesos. Objetivos: maximizar el grado de paralelismo y minimizar la comunicación entre procesos.

Localización: Los procesos se asignan a procesadores del sistema. Objetivo: ajustar los patrones de comunicación a la topología del sistema.

Escalado: Determina el tamaño óptimo del sistema en función de algún parámetro de entrada.

CAPÍTULO IV
Desarrollo

A continuación se explica en detalle cada una de las actividades realizadas de acuerdo a la metodología seleccionada para llevar a cabo la construcción del ambiente. Posteriormente se presentará la evolución de los estudios realizados a las aplicaciones DECA y EX_IN, según las metodologías recomendadas por la bibliografía para cada caso.

IV.1 Desarrollo del Middleware: Primer Prototipo

Para la creación del ambiente se consideró la utilización de los elementos que propone RAD para su uso como por ejemplo la selección del IDE Netbeans versión 5.5 para apoyar algunas áreas de desarrollo. Se debe destacar que para obtener este prototipo completo, se crearon otros muchos hasta obtener la versión definitiva, tal y como lo sugiere la metodología escogida. Se explicará el paso a paso del trabajo realizado para este primer prototipo completo clasificado en las fases de RAD.

IV.1.1 Análisis y Planificación de Requerimientos

A través de reuniones con el tutor, personal del CIDI involucrado en el proyecto, y de investigación teórica, se establecieron los requerimientos del proyecto.

Se determinó la construcción de un middleware que permitiera la ejecución de aplicaciones hechas en C y Java para un cluster de cómputo. El objetivo principal de la plataforma es permitir que los procesos laboren de manera simultánea con distintas cantidades y tipos de datos, distribuyéndolos entre las máquinas del cluster para así disminuir la carga que tendría un solo procesador. El middleware debe cumplir con los propósitos que a continuación se presentan, productos de conversaciones con el cliente:

- Hacer uso de la librería de paso de mensajes MPI.
- Implantar una herramienta de control de carga de trabajo.
- Evitar en lo posible sobrecargar de trabajo los procesadores.
- Darle continuidad a trabajos de años anteriores realizados por el CIDI referentes al tema aquí tratado pudiendo así re- usar ciertos componentes de software y bases teóricas.

Con la finalidad de reducir la subjetividad, optimizar la visión del entendimiento de los requerimientos de utilización de la plataforma y comunicar el ámbito del proyecto se recurrió a la elaboración de los casos de uso, considerándose así una herramienta muy útil en la fase de análisis de requerimientos. Como insumo inicial para la construcción de los casos de uso se creó una tabla en la cual se demuestra la relación de los usuarios del sistema con lo que ellos necesiten de éste.

Usuario/ TORQUE	¿Qué necesita del sistema?
Usuario	Ejecución del middleware.
Usuario	Validación la existencia del usuario en la base de datos.
TORQUE	Archivos insumo para que las aplicaciones paralelas facilitadas por el usuario se puedan encolar, ejecutar y archivar sus resultados.
Usuario	Observar resultados de los programas ejecutados.

Tabla 5. Relación de usuarios con el sistema. *Fuente:* Elaboración propia.

A continuación se presenta el diagrama de caso de uso general de la plataforma. Para ver otros detalles relacionados y los cuadros descriptivos de los casos de uso consultar el Apéndice G.7.

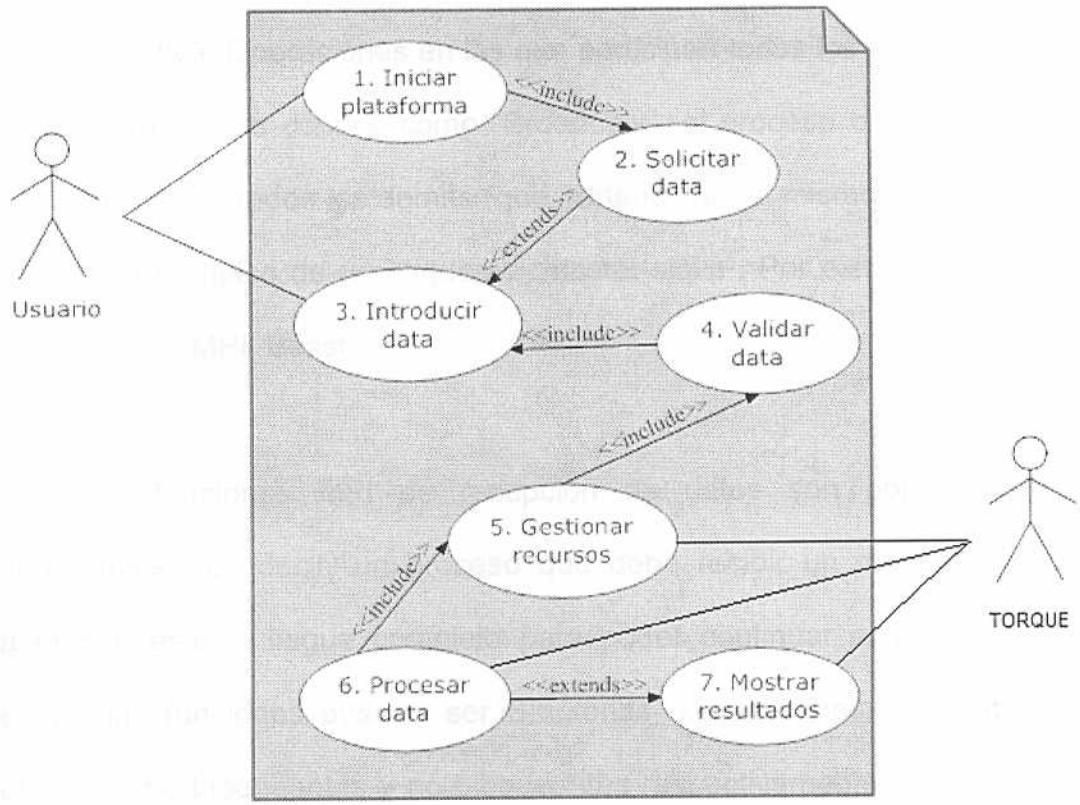

Figura 7. Diagrama de Casos de Uso de la plataforma. *Fuente:* Elaboración propia.

IV.1.2 Diseño

A continuación se muestra el diseño de los componentes involucrados para el funcionamiento de la plataforma y el de sus procesos internos.

IV.1.2.1 Diseño de la Comunicación entre los Componentes del Middleware

La comunicación MPI entre procesos puede ser de dos tipos:

- Punto a punto: el proceso "origen" conoce el identificador del proceso "destino" y envía un mensaje dirigido solo a él. Por ejemplo, se usan las funciones MPI_Send y MPI_Recv.
- Colectiva: Operaciones en las que participan todos los procesos de un operador. Se conoce como "Broadcast", el proceso origen manda el mensaje a todos los demás (que pertenezcan al mismo comunicador). Esto es típico de un esquema "master-slave". Por ejemplo, se usa la función MPI_Bcast.

Las funciones MPI de recepción de datos son por lo general "bloqueantes", es decir, un proceso que debe recibir un mensaje espera hasta que éste le llegue completo para poder continuar con su flujo. En general las funciones pueden ser Síncronas o Asíncronas, las cuales se definen como bloqueantes y no bloqueantes respectivamente. Se determinó el uso de la comunicación colectiva para la distribución de las tareas a

ejecutar, utilizándose la llamada bloqueante Scatter para que los procesos conozcan qué deben procesar.

IV.1.2.2 Diseño de Paradigma de Programación Paralela

Existen dos paradigmas de programación paralela, el paralelismo funcional y el de datos. El primero, trata diferentes tareas en paralelo de manera cooperativa, mientras que el segundo, utilizado en el middleware, establece la ejecución de una misma tarea con distintos datos.

Debido a que el número de procesos es conocido desde el principio, ya que fue determinado por el número de nodos del cluster, y a que existe una base de datos en el nodo servidor y las instrucciones de TORQUE deben ser ejecutadas desde éste, se determinó el uso del paralelismo de datos, siendo aplicable en la construcción de los archivos de extensión ".sh", los cuales sirven de insumo para la ejecución de TORQUE.

IV.1.2.3 Diseño de Archivos de Datos

El archivo "data" el cual contiene la información necesaria para que el middleware pueda trabajar, separa por bloques delimitados por comas (,), la información respectiva de cada tarea que a su vez, utiliza los espacios en blanco para comunicar cada ítem que la caracteriza. El archivo es elaborado por el usuario y debe tener la siguiente estructura:

<lenguaje><nombre><cantidad_de_procesos><naturaleza>,

<lenguaje><nombre><cantidad_de _procesos><naturaleza>

El lenguaje indica en qué lenguaje de programación está desarrollada la aplicación, sólo se contempla C, Java y algunos comandos de cónsola, el nombre es el correspondiente al nombre del archivo principal o main de la aplicación, la cantidad de procesos indica el número de procesos requeridos para la ejecución del programa, el cual por defecto es 2, debido al número de nodos del cluster y la naturaleza de la aplicación determina si se trata de un programa secuencial, o desarrollado en base a MPI. A continuación se muestra un ejemplo de dicha estructura:

```
dos ls 2 secuencial,java Amen 2 mpi,c hello 2 mpi
```

Figura 8. Estructura del archivo "data". *Fuente:* Elaboración propia.

La ruta del archivo "data" es una constante predefinida en el código del middleware, /home/mary/middleware/datos/data. Las aplicaciones que serán ejecutadas mediante el middleware deberán estar:

- Ubicadas en la siguiente ruta: /home/mary.
- Contenidas en carpetas cuyos nombres correspondan al nombre del main de la aplicación.
- Previamente compiladas y con las librerías ya configuradas.

El archivo "data_in" contiene el nombre del usuario y la contraseña del mismo, éste puede ser llenado de manera manual y también utiliza la coma (,) como medio de separación entre la data. Esta data se valida contra la ase de datos denominada APPD_db en la clase FuncionesBD.

Adicionalmente existen dos archivos generados durante la ejecución del middleware llamados "restos" y "operaciones", los cuales siguen el patrón de los dos ya descritos en relación a la separación de la data con comas, y sirven para establecer el criterio de orden de las tareas a encolar, ya que mantienen información necesaria requerida por las funciones de distribución de la data que mpiJava utiliza para tal fin.

IV.1.2.4 Diseño de la Aplicación

Con el modelo de dominio elaborado en la fase de análisis, con conceptos redefinidos, se puede elaborar un diagrama de clases con la finalidad de bajar el nivel de abstracción y contar con un insumo para la siguiente etapa, la implementación. Para ver el diagrama de clase ver el Apéndice J.

Con el diagrama de clases se determina la creación de algunos componentes de la programación, para tener una mejor visión de cómo interactúan entre ellos se definió un diagrama de secuencia.

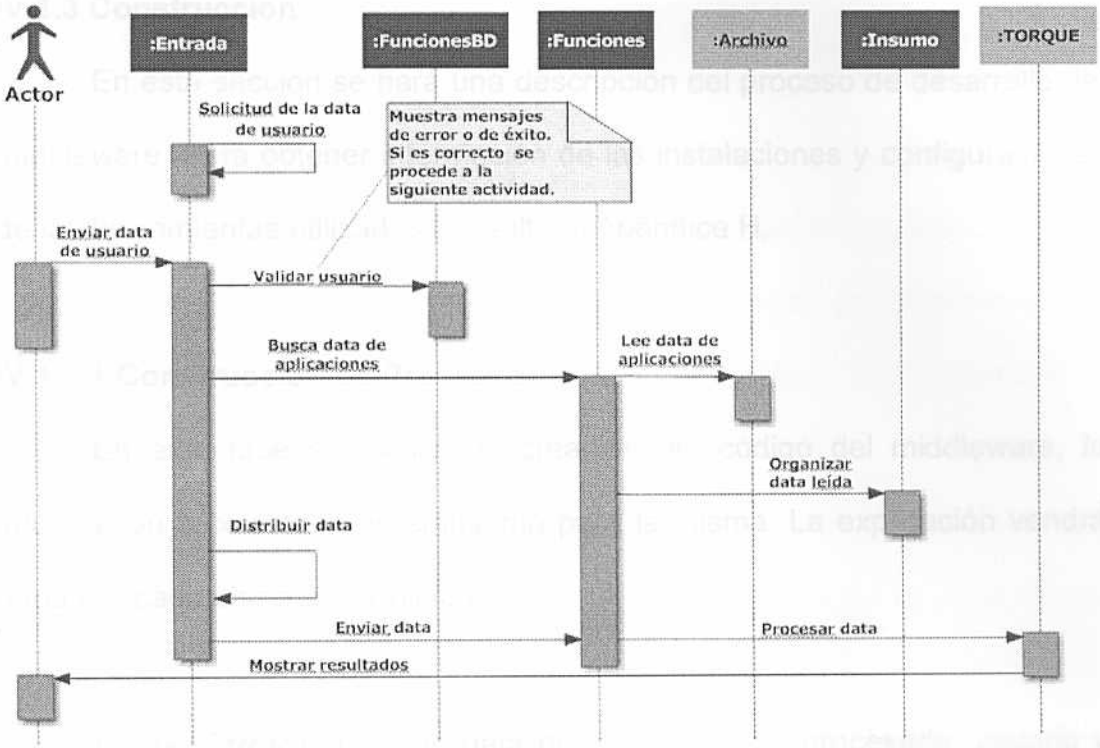

Figura 9. Diagrama de Secuencia. *Fuente:* Elaboración propia.

A continuación se describirán las clases de la plataforma:

- Entrada. Obtiene data del usuario a ser procesada, nombre e id.
- Funciones. Clase que contiene las funciones básicas a ser utilizadas por la clase Entrada.
- FuncionesBD. Clase que contiene las funciones relacionadas con la manipulación de la base de datos.
- Insumo. Clase que define al objeto encargado de almacenar la data de entrada.

IV.1.3 Construcción

En esta sección se hará una descripción del proceso de desarrollo del middleware. Para obtener información de las instalaciones y configuraciones de las herramientas utilizadas consulte el Apéndice H.

IV.1.3.1 Construcción de Procesos

En esta fase se explica la creación del código del middleware, la información recolectada es el insumo para la misma. La explicación vendrá dada por cada clase desarrollada.

Clase Entrada. Obtiene data del usuario a ser procesada, usuario y contraseña. Da inicio al proceso central, el cual se encarga de recibir, validar, organizar y distribuir la data para su procesamiento por parte de TORQUE. Presenta el procedimiento llamado operaciones, el cual elabora parámetros necesarios para la posterior distribución de los datos.

Para validar la data del usuario la clase Entrada hace un llamado a la clase FuncionesBD para así poder consultar la base de datos. Para organizar la data proveniente del archivo "data", se hace uso de la clase Insumo la cual almacena los datos leídos en un vector y luego los distribuye con la ayuda de la función de MPICH llamada Scatter, con la cual partes de un arreglo de enteros, llamado datos, que contiene los números equivalentes a los índices

de los renglones del vector con la data insumo, será dividido y transmitido a cada proceso recibiéndolo en otro arreglo llamado mis_datos.

Clase Funciones. Clase que contiene las funciones básicas a ser utilizadas por la clase Entrada. Parte de las funciones de esta clase son utilizadas tanto por el proceso central como por el esclavo. A continuación se explican las funciones y procedimientos que la conforman:

- ejecutarComando: Ejecuta las instrucciones de sistema recibidas.
- ejecutar: Llama a ejecutarComando, pasándole la instrucción determinada.
- leerArchivo: Lee el contenido de un archivo y crea un arreglo tipo String del mismo.
- eliminarDatos: Crea un arreglo de número exacto para repartir entre procesos.
- leerArchi: Lee el contenido del archivo de los programas a ser ejecutados ("data") y crea un vector del mismo, utiliza la clase Insumo para el almacenamiento de dichos datos.
- crearShs: Se encarga de la construcción de varios archivos .sh que servirán de insumo para la ejecución de TORQUE.
- crearArchivo: Crea archivo de enteros en la ruta recibida.

Clase FuncionesBD. Clase que contiene las funciones relacionadas con la manipulación de la base de datos. Es utilizada por el proceso central, para validar a los usuarios del sistema. Funciones y procedimientos:

- validarUsu: Es una función que se conecta a la base de datos ubicada en el servidor llamada APPD_db para validar los datos del usuario ingresados por el sistema. Devuelve un boolean como bandera de operación exitosa (true) o de fracasada (false).
- testDriver: Procedimiento que se encarga de verificar que el driver JDBC para MySQL esté instalado.
- getConnection: Retorna conexión a la base de datos MySQL especificada.
- consulta: Retorna flag, boolean, sobre el estatus de consulta de usuario contra la base de datos.
- executeQuery: Ejecuta la instrucción especificada y retorna el resultado, esta función permite que en un futuro se pueda incorporar otras iniciativas referentes a la edición y consulta de la base de datos.

Clase Insumo. Clase que define al objeto encargado de almacenar la data de entrada. Sus métodos son utilizados tanto por el proceso central como por el esclavo para la manipulación de la data que sirve como insumo para el procesamiento de aplicaciones a procesar de los usuarios. Funciones y procedimientos:

- ponLenguaje, ponNombre, ponProc y ponTipo son procedimientos responsables de almacenar data en variables internas.
- dameLenguaje, dameNombre, dameProc y dameTipo son funciones que devuelven el valor de la variable interna respectiva.

Adicionalmente se crearon los siguientes archivos con instrucciones de sistema:

- cd.sh: archivo de enlace, ubicado en /home/mary/middleware, se encarga de cambiar los permisos de lectura, escritura y ejecución al archivo ejec_bash.sh.
- ejec_bash.sh: Archivo con instrucciones para encolar las tareas, es creado por la clase Funciones, se ubica en /home/mary/middleware.

IV.1.4. Pruebas

Para ver las pruebas significativas realizadas para comprobar el funcionamiento de las herramientas requeridas consulte el Apéndice I, a continuación se muestran las correspondientes a la plataforma creada.

IV.1.4.1 Pruebas del Middleware Primer Prototipo

Se realizaron pruebas sencillas con la finalidad de ver las respuestas de la plataforma ante diversas situaciones. Las pruebas se basan en el criterio de la caja negra pues se enfocan en los resultados obtenidos sin

considerar el código de los mismos. Así pues se observan los mensajes obtenidos del proceso de ejecución.

A continuación se presenta una muestra de las pruebas realizadas a la primera versión de la plataforma, una de ellas es de funcionamiento y la otra de comportamiento ante los fallos. Para la prueba de funcionamiento se facilitó el archivo "data" como insumo, cuyo contenido se aprecia en la figura 8, y se observaron los mensajes de la ejecución.

```
[mary@wales middleware]$ ./script.sh

El driver para MySQL ha sido encontrado.

Conectado a la base de datos (BD):
jdbc:mysql://localhost/APPD_db?user=root&password=&holdResultsOpenOverStatementClose=t
rue.

Se ha verificado la existencia del driver JDBC y el acceso a la BD.

Tarea validar usuario correcta.

Lista de aplicaciones a ejecutarse (lenguaje nombre procesos tipo):
dos ls 2 secuencial
,
java Amen 2 mpi
,
c hello 2 mpi

Los datos recibidos por el proceso 0 son:
0
1

Los datos recibidos por el proceso 1 son:
2

/home/mary
/home/mary
266.wales.cidi.ve
267.wales.cidi.ve
268.wales.cidi.ve
[mary@wales middleware]$
```

Figura 10. Respuesta de la prueba de funcionamiento. *Fuente:* Elaboración propia.

Para la prueba de respuesta a fallos se utilizó el mismo insumo para la prueba de funcionamiento, con le diferencia que se eliminó uno de los hosts de la plataforma justo después de que el proceso central mandara a encolar las tareas a TORQUE permitiendo así observar su comportamiento, el cual fue pasivo ya que terminó la ejecución del middleware normalmente como se observa en la figura 10, pero sin dar mensaje alguno referente a la falta del nodo.

Sin embargo al ejecutar dos veces seguidas el comando pbsnodes, comando de TORQUE para conocer el estatus de los nodos, apareció lo que se aprecia en la figura 11. La respuesta de dicho comando refleja que TORQUE se percata de la ausencia del nodo. En este caso sólo le da tiempo de procesar la primera tarea de la cola, llamada ls, como se observa en la figura 12, en la cual se muestra el contenido del directorio /home/mary, en el que TORQUE tiene configurado la exposición de los resultados, con los archivos Amen.sh, ex_in.sh, ls.sh y pwd.sh como archivos de insumo para el procesamiento batch de TORQUE así como los archivos ls.sh.e456 y ls.sh.o456 que identifican los respectivos de error, dado el caso, y respuesta, procesados ya por TORQUE.

TORQUE suspende el procesamiento batch hasta que se reincorpore el nodo faltante, sin perder la lista de tareas. Efectivamente, después de que se incorpora el nodo se continúa con el procesamiento.

```
[root@germany /]# pbsnodes
germany.CIDI.ve
     state = job-exclusive
     np = 1
     properties = 192.168.1.3
     ntype = cluster
     jobs = 0/456.wales.cidi.ve
     status = opsys=linux,uname=Linux germany.CIDI.ve 2.6.17-1.2142_FC4 #1 Tue J ul 11
22:41:14 EDT 2006 i686,sessions=2318 9322,nsessions=2,nusers=2,idletime=1,
totmem=581488kb,availmem=441672kb,physmem=319352kb,ncpus=1,loadave=0.12,netload=
77598047,state=free,jobs=,varattr=,rectime=962354492
wales.CIDI.ve
     state = job-exclusive
     np = 1
     properties = 192.168.1.1
     ntype = cluster
     jobs = 0/456.wales.cidi.ve
     status = opsys=linux,uname=Linux wales.CIDI.ve 2.6.11-1.1369_FC4 #1 Thu Jun  2
22:55:56 EDT 2005 i686,sessions=2268 11488
1268729390,nsessions=4,nusers=2,idletime=5,totmem=581700kb,availmem=385712kb,physmem=3
19564kb,ncpus=1,loadave=0.20,netload=63196098,state=free,jobs=456.wales.cidi.ve,varatt
r=,rectime=962354541
[root@germany torque]# pbsnodes
germany.CIDI.ve
     state = down
     np = 1
     properties = 192.168.1.3
     ntype = cluster
     status = opsys=linux,uname=Linux germany.CIDI.ve 2.6.17-1.2142_FC4 #1 Tue Jul 11
22:41:14 EDT 2006 i686,sessions=2318
9322,nsessions=2,nusers=2,idletime=1,totmem=581488kb,availmem=441672kb,physmem=319352k
b,ncpus=1,loadave=0.12,netload=77598047,state=free,jobs=,varattr=,rectime=962354492
wales.CIDI.ve
     state = free
     np = 1
     properties = 192.168.1.1
     ntype = cluster
     status = opsys=linux,uname=Linux wales.CIDI.ve 2.6.11-1.1369_FC4 #1 Thu Jun 2
22:55:56 EDT 2005 i686,sessions=2268 11488
12687,nsessions=3,nusers=2,idletime=22,totmem=581700kb,availmem=385736kb,physmem=31956
4kb,ncpus=1,loadave=0.02,netload=63333230,state=free,jobs=456.wales.cidi.ve,varattr=,r
ectime=962354672
```

Figura 11. Respuesta ante la caída de un nodo. *Fuente:* Elaboración propia.

Figura 12. Directorio de tareas batch ante la caída de un nodo.

Fuente: Elaboración propia.

IV.1.5 Post-Actividades del Proyecto

Al finalizar la primera versión se determinaron los aspectos a considerar para ser incluidos en la segunda versión:

- Inclusión de aplicaciones paralelas desarrolladas en C que utilicen PVM ya que con la investigación teórica se verificó que TORQUE lo soporta.
- Minimizar el proceso de compilación y ejecución de la plataforma agregando nuevas configuraciones y creando los archivos con las secuencias de instrucciones necesarias para la activación de la plataforma.
- Crear una interfaz gráfica.
- Utilizar NFS para poder compartir archivos entre los nodos.
- Complementar la tolerancia a fallas que TORQUE presenta.

De igual forma en esta fase se dio inicio a la elaboración del manual de usuario que contempla la descripción del uso de la aplicación así como una lista de mensajes y comportamientos ante errores del middleware para asentar las bases de conocimiento en la organización en cuanto al uso de esta aplicación y las tecnologías que involucra.

Tras el asentamiento de los conocimientos adquiridos en relación al uso del procesamiento paralelo, de la librería de paso de mensajes MPI y mediante conversaciones con el cliente, se determinó que se hará uso del procesamiento secuencial para la construcción de los archivos ".sh", que constituyen el insumo para TORQUE, ya que el código del middleware resultaba ser muy complejo sin justificación real. Es por ello que se consideró obtener como versión final, un middleware que procesara de manera secuencial la data insumo de TORQUE para poder abarcar mayor cantidad y diversidad de tareas, empleando así MPICH y mpiJava en la ejecución de las aplicaciones paralelas suministradas por el usuario.

IV.2 Desarrollo del Middleware: Segundo Prototipo

En esta etapa se explican las nuevas incorporaciones realizadas a la plataforma: uso de NFS, PVM, interfaz gráfica, complemento a la tolerancia a fallos de TORQUE y disminución del proceso de compilación y ejecución del middleware. Las demás estructuras explicadas en el primer prototipo que no sean mencionadas acá permanecen invariables.

IV.2.1 Planificación y Análisis de Requerimientos

Tras la elaboración de las post- actividades del primer prototipo se determinó limitar el uso de mpiJava a las aplicaciones paralelas suministradas por el usuario. Se consideró la necesidad de hacer uso del NFS, motivado a que las aplicaciones a procesar por TORQUE deben estar a la disposición de todos los nodos por igual y también porque los resultados serán expresados en una única dirección para evitar la duplicidad de datos.

De igual modo se concluyó agregar la opción de que el middleware aceptara aplicaciones paralelas desarrolladas en C que hagan uso de PVM, debido a que TORQUE puede asumir dicha acción. Adicionalmente se tomaron medidas para complementar la tolerancia a fallos del middleware. En relación al modo de acceso a la plataforma, se decidió hacerla más sencilla por medio de la interfaz gráfica creada.

IV.2.1.1 Hardware

Se agregó un nuevo nodo, llamado Gibraltar, con las mismas características del nodo Wales descritas en la sección del primer prototipo, con la finalidad de utilizarlo como servidor secundario. Éste lo supliría en caso de que falle. Adicionalmente se utiliza para respaldar la data resultante del procesamiento del middleware creado.

IV.2.1.2 Ambiente de Trabajo

A continuación se describe el nuevo elemento utilizado para el ambiente de trabajo en la segunda versión del middleware:

Network File System (NFS). Protocolo que establece el servicio de archivos distribuidos en el entorno de una red. Este servicio ya estaba instalado en el cluster de cómputo usado en el CIDI. Sin embargo, se requería una nueva configuración del mismo. Se agregó la opción de compartir /home en las tres máquinas: Wales, Germany y Gibraltar.

***Parallel Virtual Machine (*PVM)**. Es un paquete de software que permite a una colección heterogénea de computadoras, con sistema operativo *UNIX*, que estén conectadas a través de una red, para ser usadas como una sola máquina paralela.

IV.2.1.3 Arquitectura del Cluster

Como ya se mencionó el cluster de cómputo empleado cuenta con la arquitectura cliente- servidor, y para esta segunda versión se consideró emplear el protocolo NFS, para el control de acceso de las máquinas del cluster a ficheros compartidos. De esta manera todos los nodos tendrán acceso a la misma información determinados archivos.

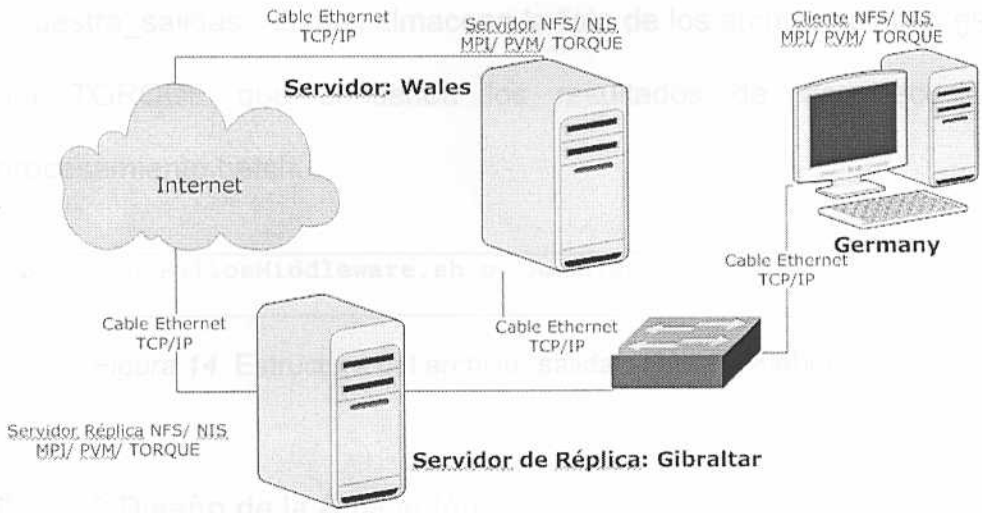

Figura 13. Nueva Arquitectura del cluster. *Fuente:* Elaboración propia.

IV.2.1.4 Arquitectura de la Aplicación

Continúa siendo cliente- servidor, pero al no incluir el procesamiento paralelo en la programación del middleware la diferenciación de procesos se observa sólo a nivel de las aplicaciones paralelas ejecutadas por TORQUE.

IV.2.2 Diseño

A continuación se mostrarán los aspectos relacionados a diseño que fueron agregados o modificados del diseño del primer prototipo.

IV.2.2.1 Diseño de Archivos de Datos

Siguiendo con la estructura de archivos planos separados por comas, en este segundo prototipo el middleware genera un archivo llamado "salida", el cual es creado por la ejecución del archivo con instrucciones de sistema

"muestra_salidas", el cual almacena la lista de los archivos planos generados por TORQUE que contienen los resultados de las ejecuciones del procesamiento batch.

```
AAAControlFallosMiddleware.sh.o, Amen.sh.out, hello_java.sh.out
```

Figura 14. Estructura del archivo "salida". *Fuente:* Elaboración propia.

IV.2.2.2 Diseño de la Aplicación

Del código en JAVA de la sección de la preparación de insumos para TORQUE del middleware creado se descartaron las funciones propias de mpiJava, como MPI.Init, MPI.Finalize, Scartter, y se modificaron otras para adaptarlas a la programación secuencial como en el caso de crearShs. Para ver el diagrama de clases de los nuevos componentes consulte el Apéndice J. A continuación se describen las nuevas clases del sistema:

- Torque. Clase que investiga estatus de TORQUE en los nodos en caso de que no se generen archivos de salidas de resultados de procesamiento batch en un tiempo predefinido.
- Salida. Clase que muestra el contenido de los archivos de resultados de TORQUE.
- CampoTexto. Clase de interfaz gráfica, usa métodos generales de las librerías awt y swing para garantizar portabilidad tanto de versiones de JDK como de sistemas operativos.

- Contenido. Clase complementaria a CampoTexto que se encarga de la manipulación de archivos planos.

IV.2.3 Construcción

En esta área se describe los pasos en la construcción de los nuevos segmentos de códigos agregados al primer prototipo, así como las modificaciones necesarias para satisfacer los nuevos requerimientos.

IV.2.3.1 Construcción de Componentes

En esta etapa se describen los nuevos componentes de la aplicación y se determinan las formas de acceso a la misma.

IV.2.3.1.1 Acceso

Existen diversas formas de obtener acceso tanto al middleware como a los resultados de su ejecución. Para poder ejecutarlo sin necesidad de una interfaz gráfica se requiere el uso de una cónsola, en la cual se debe tipear ./script.sh, siendo usuario "mary", y de esta forma se podrán observar los resultados por la misma cónsola. De igual manera, al hacer doble clic al ejecutable llamado midd ubicado en la carpeta exe del Desktop del servidor Wales, se podrá ver los resultados de la ejecución en un archivo de salida ubicado en /root/Desktop/exe, sin embargo este modo no manipula los archivos de salida, sólo muestra las tareas que fueron encoladas, el usuario

debe revisar manualmente los archivos resultantes en /home/mary, reconociéndolos por su extensión: ".sh.out".

Adicionalmente se crearon dos interfaces, una local para la ejecución y otra web, sólo informativa, las cuales serán descritas más adelante.

IV.2.3.1.2 Clases y archivos de instrucciones

Torque. Esta clase se encarga de comprobar el comportamiento de TORQUE en los nodos involucrados en caso de que después de esperar el lapso de tiempo pre- establecido para la generación de los archivos de salida por parte de TORQUE no se genere salida alguna. Funciones y procedimientos:

- ControlPBS: Método principal, gestiona control de nodos en relación a su comportamiento, es decir, según su estatus para que en caso de inactividad poder informar al administrador del sistema para tomar las medidas pertinentes.
- ejecutarComandoLocal: Ejecuta los comandos de sistemas recibidos, emplea la clase Runtime.
- crearArchivoLocal: Crea archivo en la ruta recibida, cuyo contenido se presenta en arreglo tipo char.

- leerArchivoLocal: Lee contenido de un archivo en la ruta especificada. Es una función que devuelve un entero, 1 en caso de que un nodo determinado tenga estatus "free" o 0 si tiene otro estatus diferente.
- controlGermany: Determina si TORQUE en Germany está fuera de servicio. Persiste la información relacionada en un archivo plano llamado errorPBS<nombre_host>.txt, por ejemplo en Wales en la ruta /home/mary/middleware/salidas se genera el archivo errorPBSGermany.txt.
- controlWales: Determina si TORQUE en el servidor Wales está caído, de ser así el administrador del sistema deberá establecer a Gibraltar como nuevo servidor.
- controlGibraltar: Determina si TORQUE en Gibraltar está caído. Posee el mismo comportamiento que controlGermany y Wales. Se usaría en caso de comportarse como nuevo Servidor.

Salida. Clase que se encarga de leer los archivos con los resultados de las ejecuciones batch generados por TORQUE. En caso de no obtener ninguna salida llama a la clase Torque para revisar el estatus de los nodos participantes.

- leerArchivoLocal: Lee el contenido de un archivo en la ruta especificada, imprimiendo el resultado de la lectura.

- leerArchivo: Lee el contenido del archivo llamado "salida" y crea un arreglo del mismo. Determina si TORQUE ha generado los archivos de salida.

Adicionalmente se crearon los siguientes archivos con instrucciones de sistema:

- script.sh: Se encarga de inicializar el sistema. Ubicado en /home/mary/middleware es llamado por el archivo "prueba" o por el ejecutable midd. Se puede ejecutar desde una cónsola
- antesala.sh: Ubicado en /home/mary/middleware, es ejecutado desde la clase de interfaz local llamada Contenido, para poder cambiar el usuario y los permisos del archivo "data_in" creado desde ésta con usuario "root" para luego ejecutar el archivo prueba como usuario "mary".
- prueba: Ubicado en /home/mary, es ejecutado desde el archivo "antesala.sh" para ejecutar el archivo script.sh y almacene el resultado en un archivo plano llamado final.
- muestra_salidas: Ubicado en /home/mary, se encarga de ejecutar las clases que procesan las salidas de TORQUE, mediante la llamada a la ejecución de la clase Salida.
- en: Sirve de enlace para la ejecución del middleware sin interfaz gráfica. Es llamado por el programa desarrollado en c llamado midd.

IV.2.3.1.3 Interfaz

Existen dos tipos de interfaz. Una sencilla local que consta de una ventana en la cual se puede ingresar el nombre del usuario y contraseña para luego observar los resultados de la ejecución en la misma ventana, y otra interfaz web informativa, con la cual se podrá observar el resultado de la última ejecución del middleware.

Esta última interfaz fue creada usando PHP y html, la cual fue probada en el laboratorio mediante la LAN, usando apache como servidor.

Interfaz Local

Esta interfaz emplea las clases awt y swing, sin intervención de paquetes propios de IDESs determinados, esto debido a que el JDK empleado no posee muchas alternativas en cuanto a la interfaz, sin embargo esta particularidad mejora la portabilidad de la aplicación. A continuación se muestran los componentes:

CampoTexto. Clase encargada de generar la interfaz local. Se compone de:

- LeerArchivoLocal: Lee el contenido de un archivo en la ruta especificada y lo muestra en un JTextArea de la ventana de la interfaz.
- ManejadorCampoTexto: Es una clase interna privada para el manejo de eventos.

- CampoTexto: Es el constructor de la clase y en él se definen los elementos gráficos de la ventana.

Contenido. Clase que complementa a la clase CampoTexto en relación a la manipulación del archivo "data_in", utilizado para almacenar a los usuarios del sistema. Componentes:

- ejecutarComandoLocal: se encarga de ejecutar el archivo antesala.sh, el cual contribuye con la inicialización de la ejecución del middleware.
- crearArchivo: crea un archivo de enteros en la ruta recibida. Sirve para almacenar la data del usuario (nombre y la clave del usuario), en el archivo plano "data_in".

Interfaz Web Informativa

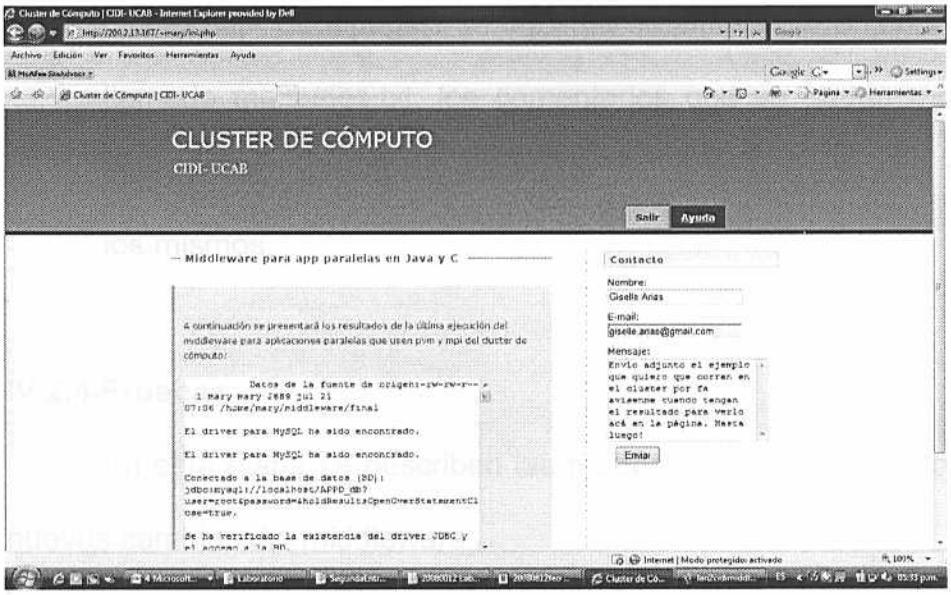

Figura 15. Interfaz Web Informativa. *Fuente:* Elaboración propia.

Se creó un directorio llamado public_html en /home/mary, y se habilitó la opción de apache para crear directorios para usuarios para poder acceder a esta interfaz perteneciente al usuario "mary" siempre y cuando el servicio de apache esté disponible. Mediante cualquier nodo de la LAN perteneciente al cluster se visualiza colocando en el navegador lo siguiente: http://200.2.13.167/~mary, mientras que en la misma máquina Wales se coloca http://localhost/~mary para garantizar el acceso. Esta interfaz muestra el contenido del archivo plano llamado final ubicado en /home/mary/middleware, junto con los datos relacionados a éste como la hora y fecha de creación. Además de la plantilla web utilizada, descargada de manera gratuita de internet, está conformada por los siguientes archivos:

- index.php: Contiene la página principal de la interfaz en la cual se muestra la última ejecución del middleware.
- correo.php: Esta página se encarga de almacenar en un archivo plano llamado mensajes.txt, los comentarios que se hagan mediante esta interfaz para que el administrador los considere y pueda responder a los mismos.

IV.2.4 Pruebas

En esta etapa se describen las pruebas realizadas relacionadas a los nuevos cambios del middleware.

IV.2.4 .1 Verificación de NFS

A continuación se mostrarán los pasos seguidos para verificar la debida instalación y configuración del NFS en el cluster de cómputo. Primero en el servidor se revisan los archivos /etc/exports, /etc/hosts.deny, /etc/hosts.allows y /etc/hosts, con la finalidad de realizar las ediciones pertinentes para poder establecer la comunicación entre los nodos del cluster. Mientras que del lado del cliente basta con verificar los archivos /etc/hosts y /etc/fstab.

Para ambos casos se debe revisar que el servicio esté activo, en caso de que haya dudas puede aplicarse desde la cónsola como usuario "root" el comando: /sbin/service nfs restart o /sbin/service nfs start.

```
root@wales:/
Archivo  Editar  Ver  Terminal  Solapas  Ayuda
[root@wales ~]# insmod /lib/modules/2.6.11-1.1369_FC4/kernel/fs/nfs/nfs.ko
insmod: error inserting '/lib/modules/2.6.11-1.1369_FC4/kernel/fs/nfs/nfs.ko': -
1 Unknown symbol in module
[root@wales ~]# nfs man
bash: nfs: command not found
[root@wales ~]# cd ..
[root@wales /]# /sbin/service nfs start
Inicio de los servicios NFS:
[root@wales /]# /sbin/service nfs restart
Cerrar NFS mountd:                                         [FALLÓ]
Cerrar el demonio NFS:                                     [FALLÓ]
Apagando cuotas NFS:                                       [FALLÓ]
Apagar los servicios NFS:                                  [  OK  ]
Inicio de los servicios NFS:                               [  OK  ]
Iniciando cuotas NFS:                                      [  OK  ]
Inicialización del demonio NFS:                            [  OK  ]
Inicialización de NFS mountd:                              [  OK  ]
[root@wales /]#
```

Figura 16. Ejemplo de Reinicio del Sistema. *Fuente:* Elaboración propia.

Para montar una carpeta, además de incorporarla al archivo fstab del lado del cliente, se debe utilizar el comando mount. Por ejemplo: mount 192.168.1.1:/home /home, de esta manera el nodo cliente que ejecutó el comando establece el enlace con el servidor, en este caso 192.168.1.1, pudiendo entonces compartir la carpeta /home.

IV.2.4.2 Verificación de PVM

Para que el middleware admitiera la ejecución de aplicaciones paralelas con PVM fue necesario incorporar un segmento en el código de la clase Funciones, específicamente en la función crearShs, con la finalidad de que la construcción de este archivo esté acorde al caso. A continuación se muestra un ejemplo de la ejecución mediante TORQUE de una aplicación desarrollada en C que usa PVM, al aplicar el comando qsub –l nodes=2 problem.sh desde la cónsola.

#! /bin/bash pwd cd .. cd .. cd /usr/share/pvm3/bin/LINUX ./problem	/home/mary hello, I'm the master with tid 262146 running on machine wales.CIDI.ve The array of 10 elements contains 8 prime numbers. Parallel Time : 0s 98028us The array of 10 elements contains 8 prime numbers. Sequential Time : 0s 25us Speedup : 0.000255029 Efficiency : 0.000127515

Figura 17. Ejemplo de uso de aplicación de PVM con TORQUE, archivo "problem.sh" y "problem.sh.o230". *Fuente:* Elaboración propia.

IV.2.4.3 Pruebas de control de fallos

En esta sección se realizaron dos tipos de pruebas, la primera tiene que ver con el módulo que consulta a Torque sobre su estatus para cada nodo en caso de que el sistema no encuentre archivos de salida en un tiempo predefinido. En este caso Germany no está activo, se ejecuta la aplicación y se obtiene un mensaje de error, al consultar el manual de usuario del sistema el usuario se entera que debe revisar el directorio /home/mary/middleware/salidas y buscar por los archivos de error generados, al hacerlo éste encuentra el archivo errorPBSGermany.txt. Ver Apéndice K para ver continuación de la prueba.

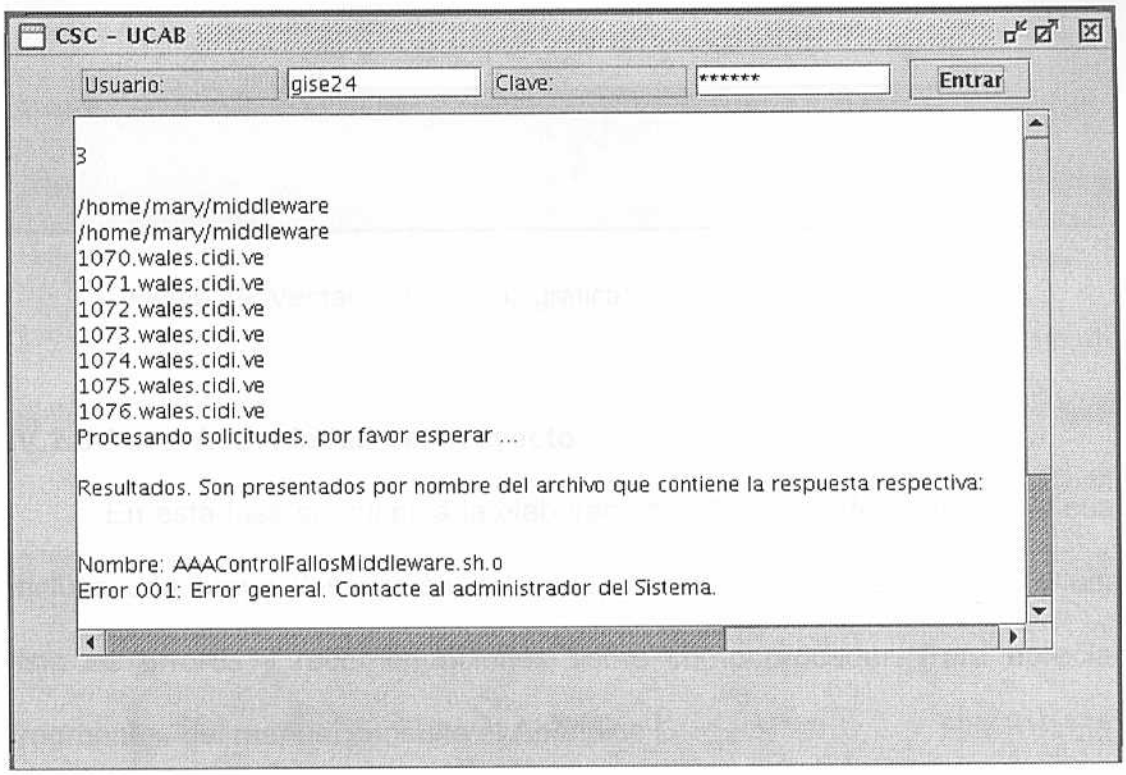

Figura 18. Prueba del módulo de TORQUE. *Fuente:* Elaboración propia.

IV.2.4.4 Pruebas de Interfaz

A continuación se muestra cómo se ve la interfaz gráfica.

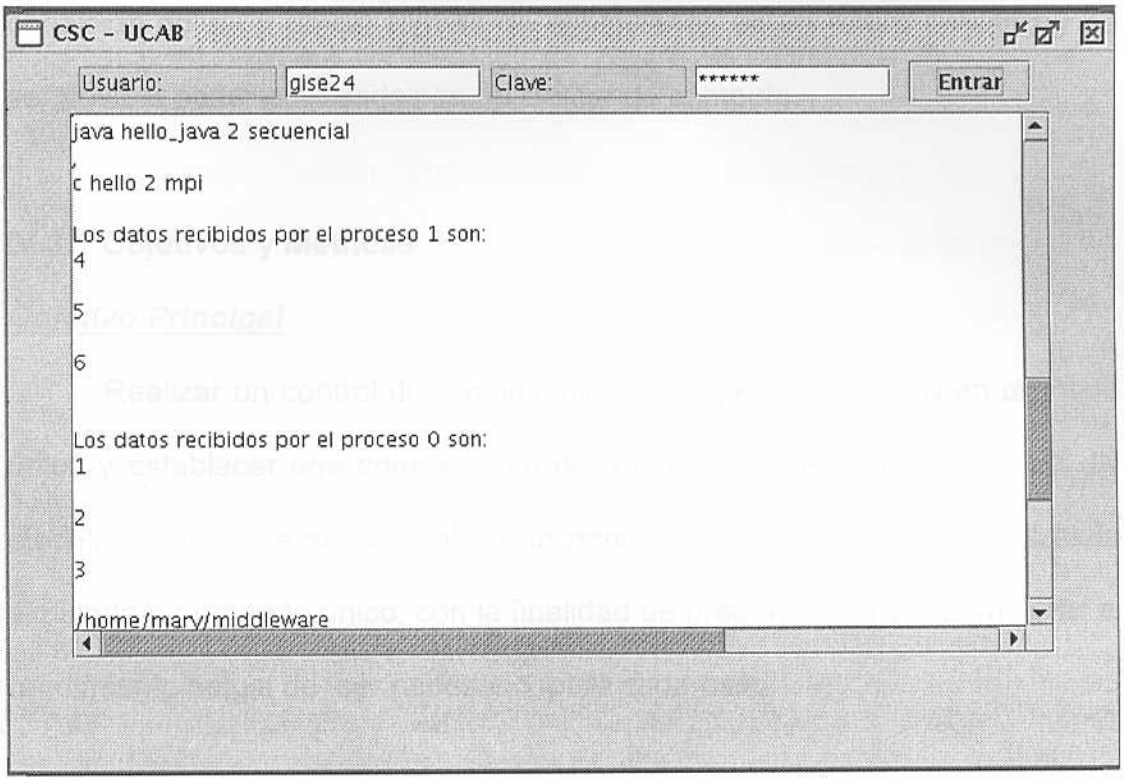

Figura 19. Ventana de interfaz gráfica. *Fuente:* Elaboración propia.

IV.2.5 Post- Actividades del Proyecto

En esta fase se culmina la elaboración del manual de usuarios, el cual incluye una breve descripción del middleware, instrucciones de uso y una lista de errores y recomendaciones sobre cómo proceder. Para apreciar fragmentos del manual consulte el Apéndice L.

V.3 Desarrollo de Estudios de Rendimiento

A continuación se presentan los estudios de rendimiento realizados a dos de las aplicaciones pertenecientes al CIDI, con la finalidad de conocer su condición actual y ubicar formas de optimizarlas en un futuro cercano y de ser posible poder emplearlas en un cluster de cómputo.

IV.3.1 Objetivos y Métricas

Objetivo Principal

Realizar un control de rendimiento de la aplicación DECA en un nodo único y establecer una comparación de rendimiento de la aplicación EX_IN ejecutada de forma secuencial en un nodo único y luego de forma paralela simulada en un nodo único, con la finalidad de proponer formas de mejorar el rendimiento actual de ser necesario, para cada caso.

Objetivos Específicos

- Seleccionar las métricas y clasificarlas.
- Seleccionar las técnicas de evaluación a utilizar.
- Determinar parámetros y factores de las mediciones.
- Determinar el ambiente de hardware y software para realizar las mediciones.

Métricas

- Se utilizará la métrica asociada a la velocidad, tiempos de respuesta, para determinar qué tan rápido se ejecutan las aplicaciones.

Clasificación de las Métricas:

LB (más bajo mejor): Se seleccionó esta clasificación ya que mientras más pequeño sea el tiempo de respuesta será mejor las conclusiones ya que lo que se necesita de estas aplicaciones es que los resultados de las mismas se obtengan en el menor tiempo posible.

IV.3.2 Parámetros y Factores de las Mediciones

Los parámetros son los elementos constantes durante la evaluación mientras que los factores son los variantes, a continuación se identifican ambos aspectos según los estudios a realizar.

Parámetros del sistema

Hardware

Nodo único para el control de rendimiento de DECA:

AMD Athlon 1700+, 1467 Mhz, RAM: 2 GB

Ejecución secuencial y simulada paralela de EX_IN:

AMD Athlon, 1.10 Ghz, RAM: 368 MB

Software

Nodo único para el control de rendimiento de DECA:

Sistema Operativo Linux Debian 6.0.

Netbeans 5.5, JDK 1.6.0.

Librerías adicionales:

Jdom, Jai i586 para Linux, MySQLConnector 5.0.

Nodo único para EX_IN:

Gcc, compilador de C para Linux.

Parámetros de carga de trabajo

Además de las propias aplicaciones a evaluar se cuenta como parámetros de carga los datos de entrada de cada una de ellas.

EX_IN: archivos de entrada y otras constantes de la aplicación.

DECA: archivo de entrada con datos, archivo .xml o imágenes a procesar.

Factores

Para el estudio de EX_IN, se considera el tipo de la aplicación, si es secuencial o simulada paralelo así como la existencia de un bucle que cuente iteraciones en el código fuente.

Para DECA, posible utilización de otras aplicaciones durante las pruebas como Internet Explorer o algún editor de texto y variación de imágenes de fondo y principal.

IV.3.3 Instalaciones y Configuraciones

Para poder realizar las mediciones de DECA fue necesario instalar y configurar algunos componentes inexistentes en el cluster, las cuales se describen a continuación:

El JDK empleado fue el 6.0 el cual se descargó e instaló de la misma manera en la que se hizo el 1.4 en el cluster de cómputo. Netbeans 5.5 y su componente C/C++ se descargaron e instalaron, descomprimiendo los archivos correspondientes para luego seguir las instrucciones de un wizard de instalación. Para que DECA compilara correctamente en Netbeans fue necesario especificar el nombre de la plataforma: Java HotSpot(TM) Client VM 1.6.0_03-b05, así como el del JAI, que fue copiado en la carpeta jre del JDK. También se especificó el JDK mencionado, el cual fue agregado al proyecto mediante las siguientes opciones del menú superior de Netbeans: Tools->Library Manager-> Add JAR/Fólder y se especificó la ruta. De igual forma para las otras librerías: Tools->Library Manager, se crearon dos nuevas librerías con el nombre de JDOM y MySQLConnector indicándoles las rutas respectivas. Para ver las mediciones y análisis de los programas ver el Apéndice M.

IV.4 Desarrollo de Estudio de Paralelización de DECA

Recordando las fases de la programación paralela (Descomposición Funcional, Partición, Localización y Escalado) que pueden ser consultadas en el capítulo del Marco Metodológico, se presenta un estudio teórico cuyos resultados están enmarcados en las mismas, reflejando así una forma de convertir la aplicación DECA desarrollada de manera secuencial en una paralela. A continuación se muestra una breve reseña de la aplicación, dividida en planteamiento, esquema general y análisis de la aplicación para así poder obtener una mayor comprensión de ésta.

IV.4.1 Planteamiento

El sistema DECA surge como requerimiento para el proyecto PISACOTA 1, el cual tiene como objetivo realizar la tarea de adquisición, análisis, procesamiento, modelaje, simulación y visualización del tráfico automotor en una vía determinada.

DECA, tiene como objetivo principal detectar el tipo y la cantidad de vehículos presentes en una vía en un instante a partir de una imagen digital. Debido a que este sistema fue elaborado de forma monolítica en el lenguaje de programación JAVA, y a que la utilidad que el CIDI le daría consistiría en el procesamiento de múltiples imágenes en un instante determinado resulta vital importancia realizar un estudio sobre las posibilidades que tiene dicha aplicación para ser ajustada al procesamiento paralelo. De ser así se

recomendaría una forma de hacerlo con la finalidad de ponerla en práctica y así colocarla en funcionamiento en el cluster de cómputo del centro.

IV.4.2 Esquema General de la Aplicación

Módulo de Administración de Canales de Procesamiento de Imágenes (CPI): Brinda al usuario la posibilidad de crear, iniciar y configurar cada CPI.

Módulo de Configuración: Ofrece la posibilidad al usuario de establecer los parámetros correspondientes a cada CPI. Este módulo se encarga de asociar dichos parámetros con los CPI presentes en el sistema a través de una interfaz gráfica.

Módulo Manejador de Entradas: El manejador de entradas se encarga de proveer de imágenes a cada *CPI*.

Manejador de Salidas: Un manejador de salida al igual que uno de entrada, se encarga de proveer un nivel de independencia entre cada *CPI* y el modo en que el usuario desea que le sean presentados los resultados.

Módulo Manejador de Segmentación: Este módulo contiene los métodos de segmentación disponibles al usuario para el funcionamiento del sistema. DECA únicamente propone la técnica de segmentación basada en regiones. Sin embargo, el sistema tiene la capacidad de usar otros métodos implementados por el usuario.

Módulo de Procesos: Se encarga de realizar el procesamiento de la imagen.

IV.4.3 Análisis

Esta sección de análisis coincide con la fase de programación paralela denominada Descomposición Funcional ya que es aquí donde se identifican las funciones críticas para el procesamiento de la data, que serían entonces el foco para la transformación del código secuencial a paralelo.

Motivado a que el módulo de Procesos es el destinado a la ejecución del procesamiento de imágenes, el estudio se enfocará en determinar cómo esta área del código del sistema podría ser transformada para ejecutarse de manera paralela, esto con la finalidad de ser elaborado en un futuro y así aplicar alguna de las métricas de rendimiento para determinar la efectividad del cambio.

El módulo de Procesos cuenta con las siguientes clases:

Clase Proceso: Clase abstracta que define los atributos y métodos básicos de todo CPI, atributos:

· *identificador*: La cadena que identifica la configuración que está siendo ejecutada por el Proceso.

· *configuracion*: El objeto Configuracion que contiene los parámetros para el procesamiento.

· *input*: El manejador de entrada (Entrada).

· *output*: El manejador de salida (Salida).

· *segmentador*: El segmentador a ser utilizado en el procesamiento. Este objeto no es creado al momento de crear el objeto Proceso, se deja que el programador lo haga cuando sea necesario.

· *verbose*: Es de tipo boolean y le indica al CPI si el usuario desea hacer impresiones adicionales para dar más información.

Proceso define el método void run(), que se encarga de realizar el procesamiento. También define los métodos void print(...) y void println(...), que están relacionados con el valor del atributo verbose, realizando impresiones en la salida estándar cuando su valor sea afirmativo.

Clase FabricaProceso: Provee el método *construir(...)*, como el tipo de *CPI* a utilizar no es parte de la configuración, se debe especificar el tipo deseado, por lo que se debe indicar en el productor cual será la *clave* del *CPI,* esta *clave* es una cadena de caracteres alfanuméricos, sin símbolos, caracteres especiales o espacios en blanco.

Clase ProcesoVacio: Ésta no realiza ningún procesamiento, sólo muestra información sobre la configuración que le fue asociada. Su *clave* es "*void*".

Clase ProcesoEstandar: Realiza el procesamiento estándar del sistema. Su *clave* es "standard". Las clases utilitarias que fueron

desarrolladas simplificaron la implementación de este *CPI*, la clase *Operaciones* provee todos las operaciones necesarias para que la implementación fuera lo más sencillo posible.

Luego de conocer la estructura básica del módulo se identificó el método clave en el procesamiento de imágenes, el cual será descrito a continuación:

Método procesar: Es el método que pone en marcha el procesamiento de las imágenes. En general, el procesamiento de imágenes está compuesto de tres etapas: Preprocesamiento, Procesamiento y Postprocesamiento

Una tarea DECA (o Proceso), es básicamente igual, agregando dos etapas: Adquisición de la imagen, Preprocesamiento, Procesamiento, Postprocesamiento y Entrega de resultados.

Este método debe llevar el control de todo el procesamiento, y debe asegurar que se va a pasar por todas esas etapas. Varía según la configuración porque cada proceso es diferente en lo que hace, por ejemplo, para la configuración de Realidad Virtual se aplicaron resta de imágenes, luego se segmentó usando cierto método de segmentación, aplicando ciertos filtros. Así como la configuración es flexible para cada cámara, el

procesamiento también, ya que por ejemplo se podría tener una cámara a la que le pegue mucha luz y otra a la que no, y respectivamente 2 procesos, uno para casos en los que hay mucha luz y otro procesamiento cuando no.

```
protected boolean procesar() {

    //hacer sucesivos update backgrounds y mostrar cada iteracion....
    //la primera completa y luego si solo los resultados
    JFrame unFrame = new JFrame();
    unFrame.setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
    unFrame.setSize(760,600);
    muestra = new GaleriaDECA(GaleriaDECA.PIZARRA);

    this.verbose = true;

    System.out.println("Cargando las imagenes...");
    //1 - ADQUISICION DE LAS IMAGENES (segun la conf. del input)

    do{
        PlanarImage actualOriginal = input.getImagenPlanarImage(Entrada.IMAGEN_ACTUAL);

         if (verbose)
            muestra.addTabImagen("Actual", actualOriginal.getAsBufferedImage());

        PlanarImage fondoOriginal = input.getImagenPlanarImage(Entrada.IMAGEN_FONDO);

        if (verbose)
        muestra.addTabImagen("Fondo", fondoOriginal.getAsBufferedImage());

        PlanarImage comboResta = Operaciones.comboSubtraerImagenes(actualOriginal,fondoOriginal);

         if (verbose)
             muestra.addTabImagen("Resta", comboResta.getAsBufferedImage());

        this.updateBackground( actualOriginal, fondoOriginal, comboResta);


        this.verbose = false;

    } while(input.haySiguiente());

    unFrame.add(muestra);
    unFrame.setVisible(true);

    return false;
}
```

Figura 20. Código Secuencial del método Procesar en DemoUpBack.

Fuente: Trabajo de Grado "Sistema de detección de tipo y cantidad de vehículos presentes en una imagen digital".

CAPÍTULO V
Resultados

Con base en lo presentado en el capítulo de Desarrollo en relación al middleware, a las mediciones realizadas para llevar a cabo los estudios de rendimiento de las aplicaciones EX_IN y DECA así mismo con el estudio para llevar a código paralelo a DECA se obtuvieron los siguientes resultados:

1. Se realizó una investigación con base teórica en la que se analizan los componentes, ventajas y desventajas más importantes de dos manejadores de colas, TORQUE y OpenPBS. De esta investigación, mediante cuadros comparativos, se seleccionó TORQUE como herramienta a instalar, configurar y utilizar para el procesamiento batch que usa el middleware por ser una de las herramientas más actualizadas, software libre y por contar con numerosa documentación.

2. Para el funcionamiento del ambiente de programación paralelo se realizaron las siguientes instalaciones y configuraciones:

   Jdk1.4, versión de la máquina virtual de JAVA recomendada para ser utilizada con mpiJava.

   MPICH, distribución de la librería de paso de mensajes MPI para aplicaciones paralelas desarrolladas en C.

mpiJava, distribución de la librería de paso de mensajes MPI para aplicaciones paralelas desarrolladas en JAVA.

TORQUE, gestor de recursos que proporciona control sobre trabajos batch.

3. El middleware resultante puede procesar algunos comandos de sistema, aplicaciones tanto paralelas como secuenciales (aprovechando el procesamiento batch mediante TORQUE), desarrolladas en lenguaje C y JAVA, en el caso de las paralelas utilizando MPI y PVM como valor agregado.

4. El middleware cuenta con un manual de usuarios que contiene la explicación del uso de la aplicación y una lista de mensajes de errores con significado y opciones para resolverlos.

5. Se implementó una interfaz local con el uso exclusivo de las librerías awt y swing, debido a que el JDK usado al no ser reciente tiene opciones limitadas en cuanto a interfaz se refiere, sin embargo esta característica permite optimizar la portabilidad del middleware.

6. Se configuró e implementó un servicio NFS, para compartir directorios entre los nodos del cluster y un servicio apache para un usuario

determinado en la que se disfruta de una interfaz web, como valor agregado.

7. Se realizaron las mediciones y el análisis de las mismas para la comparación de rendimiento de la aplicación secuencial EX_IN entre sus dos versiones, considerando parte de los diversos factores que podrían alterar su rendimiento.

8. Se realizaron las mediciones y el análisis de las mismas para el control del rendimiento de la aplicación DECA, en un ambiente similar al utilizado en su creación, considerando parte de los diversos factores que podrían alterar su rendimiento.

9. Se realizó el estudio de la paralelización de la aplicación llamada DECA, el cual resultará ser muy útil al momento de materializar la actividad de la paralelización ya que al usar las fases de la programación paralela el tamaño y complejidad del programa se minimizan convirtiendo a este estudio en una guía para el programador que será responsable de llevar a cabo esta actividad.

CAPÍTULO VI
Conclusiones y Recomendaciones

VI.1 Conclusiones

Tomando como base los resultados ya presentados se obtuvieron las siguientes conclusiones:

- La realización de este trabajo de grado permitió aplicar los conocimientos adquiridos durante la formación académica y asimilar nuevas tecnologías, apreciando la importancia del levantamiento de información y la manipulación de la misma haciendo uso de la metodología más apropiada y considerando los recursos disponibles.
- Aunque el hardware disponible haya sido limitado, debido a la velocidad de procesamiento y cantidad de nodos, se logró implementar un middleware con diversas utilidades.
- Se utilizaron herramientas open source disponibles en Internet como el Sistema Operativo Fedora Core 4, librerías de programación, TORQUE, MPICH, JDK 1.4, mpiJava.
- El uso de NFS, en este caso, para la ejecución batch de aplicaciones paralelas y secuenciales resulta ser necesario para tener una sola copia común para todos los nodos, tanto de la data y aplicación a procesar como el resultado a obtener.

- Para el estudio realizado de la aplicación EX_IN, en la sección que se aplica Anova y el de Análisis de Varianza, se demuestra que el factor determinante en las diferencias de rendimiento es el que especifica si la aplicación es secuencial o paralela simulada, independientemente de que si deban tener un procesamiento extra equivalente al bucle for definido.
- Para el estudio realizado de la aplicación EX_IN, en la comparación de la forma con y sin bucle tanto secuencial como simulado paralelo, el valor de la F o variación entre grupos es de 2796,772 y 1875,711 respectivamente. Para determinar si los resultados tienen un valor significativo (Probabilidad < 0,05) el valor de F necesita ser al menos 4,098, como la diferencia es tan grande se puede asegurar que existe una diferencia significativa entre la aplicación secuencial y simulada paralela con y sin el bucle for.
- Para los casos estudiados de EX_IN, se obtuvo una varianza pequeña al igual que la suma de los cuadrados, con lo cual se aprecia la concentración y consistencia de los datos y por tanto de las mediciones. De igual forma en el estudio de DECA al obtener un SST pequeño se evidencia la consistencia de los valores obtenidos.
- En el estudio de la aplicación DECA se concluyó que el factor determinante en la diferencia de rendimiento es el involucrado a la variación de programas ejecutándose mientras se corre la aplicación.

- En el estudio de paralelización de la aplicación DECA, para el módulo de Procesos, considerando la fase de Partición, se concluyó que el uso la librería de paso de mensajes mpiJava junto con NFS deben conformar el esquema de comunicación entre procesos, así mediante la fase de Localización, se identificó la función Scatter para distribuir la data entre los nodos y la función Gather para la recolección de resultados.
- Para determinar si una aplicación debe requerir del procesamiento paralelo se debe determinar el tipo de manipulación de la data insumo y el cálculo de los resultados.
- No todas las aplicaciones secuenciales deben transformarse a paralelas.
- El impacto de agregar al middleware otro lenguaje de programación para el procesamiento paralelo no será significativo siempre y cuando exista la librería para el procesamiento paralelo que lo soporte.

VI.2 Recomendaciones

A continuación se presentan las recomendaciones pertinentes para asegurar el uso de buenas prácticas mejorar el trabajo realizado:

- En caso de que la interfaz web informativa del middleware se publique en internet se recomienda agregar un módulo de validación de

usuarios para garantizar la discreción de los resultados de las aplicaciones ejecutadas en batch.

- Se recomienda hacer pruebas con un JDK más reciente y emplear una versión más actual para el estándar MPI para JAVA, como mpj, así como otras pruebas de aplicaciones desarrolladas en FORTRAN adaptadas a MPI.
- Para el uso de MPICH y TORQUE se recomienda ejecutar las pruebas de verificación expresadas en el Apéndice I, haciendo uso de los comandos referentes para ello allí mencionados.
- Se recomienda crear la versión paralela de EX_IN con MPICH y ejecutarla mediante el middleware, de la misma manera se recomienda seguir las fases de la programación paralela para esta actividad.
- De igual manera para la versión paralela de DECA, se recomienda considerar la fase de Escalado para optimizar su tamaño y así su rendimiento.
- Para evitar advertencias al momento de instalaciones y configuraciones que podrían ser confusas para el usuario es recomendable mantener la hora y fecha del sistema actualizada.
- Es recomendable que en un cluster que use un gestor de trabajos batch, la máquina que sirva de servidor sea la única conectada a

Internet, los nodos no deberían estar en esa exposición, asimismo se debe revisar las colas y nodos disponibles de manera periódica.

- Antes de instalar cualquier librería de paso de mensajes así como cualquier otra aplicación se debe verificar los pre-requisitos para garantizar un óptimo funcionamiento de la nueva instalación.
- Para el middleware creado se recomienda automatizar la carga de los archivos a ser procesados, así como realizar un estudio para determinar las acciones necesarias para también automatizar la tolerancia a fallos, específicamente en relación a la transición de un servidor a otro.
- Se recomienda emplear la fase de programación paralela denominada Escalado, para así realizar los estudios necesarios de las aplicaciones paralelas que se deseen ejecutar con la finalidad de determinar la cantidad óptima de nodos del cluster a emplear.
- Incrementar el uso de nombres descriptivos tanto para archivos como variables del middleware.

BIBLIOGRAFÍA

- Uzcátegui, Mariana. Informe # 1 Middleware de un Cluster. Centro de Investigación y desarrollo de Ingeniería –CIDI- Universidad Católica Andrés Bello –UCAB-Montalbán, Caracas. [2006].

- Tanembaum, Andrew S. (1996). *Sistemas Operativos Distribuidos*. México: Prentice- Hall Hispanoamericana, S.A.

Referencias Electrónicas

- Bosque, José Luis. *Tema 4: Middleware y Programación Paralela de Clusters*. Laboratorio de Arquitecturas Avanzadas de Computadores 5º Ing. Informática, [en línea]. Disponible en: http://dac.escet.urjc.es/docencia/LAAC/LAAC_Tema4.pdf. [2008, 30 de enero].

- Mikel A. K. y Buytaert K. *Cap. 44 Live Linux CD*. [En línea]. Disponible en: http://www.wikilearning.com/tutoriales/manual_funciones/busqueda/1. [2006, 27 de febrero].

- Baker Mark, Carpenter Bryan, Fox Georey, Hoon Ko Sung y Lim Sang. *mpiJava: An Object- Oriented Java interface to MPI.* School of Computer Science University of Portsmouth, Southsea, Hants, UK, NPAC at Syracuse University Syracuse, New York, USA. Disponible en: http://ipdps.cc.gatech.edu/1999/java/baker.pdf. [1999].

- Alonso, José Miguel. *Programación de aplicaciones paralelas con MPI (Message Passing Interface).* Facultad de Informática UPV/EHU, [en línea]. Disponible en: http://www.sc.ehu.es/acwmialj/edumat/mpi.pdf. [1997, 13 de enero].

- *Terminología Apunte sobre los términos "computación e informática".* ACIMED. Disponible en: http://bvs.sld.cu/revistas/aci/vol2_3_94/aci09394.htm. [1994, septiembre- diciembre].

- Bernal C., Iván, Mejía N., David y Fernández, A. Diego. *Desarrollo de aplicaciones paralelas para clusters utilizando mpi (message passing interface).* Escuela Politécnica Nacional, [en línea]. Quito. Disponible en: http://clusterfie.epn.edu.ec/clusters/Publicaciones/HTML/articulo2.htm.

- Rodríguez, David. *Computación Paralela.Grids & e- Ciencia. Curso Postgrado CSIC*. Consejo Superior de Investigaciones Científicas Instituto de Fìsica de Cantabria, IFCA (Centro Mixto CSIC-Universidad de Cantabria), [en línea]. Santander: España. Disponible en: http://209.85.215.104/search?q=cache:7fgEeGZeeVQJ:grid.ifca.es/Curso_Grid/Presentations/MPI-2006-final.ppt+David+Rodr%C3%ADguez.+Consejo+Superior+de+Investigaciones+Cient%C3%ADficas+Instituto+de+F%C3%ACsica+de+Cantabria,+IFCA&hl=es&ct=clnk&cd=2&gl=ve. [2006, junio].

- Wikipedia *Network Information System*. Disponible en: http://es.wikipedia.org/wiki/Network_Information_Service. [2008].

- Soto, Miguel Alejandro. Protocolos TCP/IP. Disponible en: http://usuarios.lycos.es/janjo/janjo1.html. [2008].

- Wikipedia. *Categoría: Paradigmas de Programación*. Disponible en: http://es.wikipedia.org/wiki/Categor%C3%ADa:Paradigmas_de_programaci%C3%B3n. [2008].

- Wikipedia. *Ley de Amdahl*. Disponible en: http://es.wikipedia.org/wiki/Ley_de_Amdahl. [2008].

- Trivoli Workload Scheduler LoadLeveler library. *Glossary*. Disponible en: http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp?topic=/com.ibm.cluster.loadl.doc/loadl331/am2ug30331.html. [2008].

- Innocente, Roberto. *MPI Performance of a PC cluster at the ICTP*. Abdus Salam ICTP/Sissa, [en línea]. Disponible en: http://hpc.sissa.it/1st/index.html. [1999, 22 de marzo].

- Diaz, Gilberto. *Portable Batch System*. Centro de Cálculo Científico Universidad de los Andes, [en línea]. Mérida: Venezuela. Disponible en: http://www.cecalc.ula.ve/HPCLC/slides/day_10/PBS.pdf. [2003, octubre- noviembre].

- Universia. Boletines. Disponible en: http://servicios.universia.edu.ve/rediseno/contenidos/sencillos/detalle.php?id_content=4839. [2008].

Tesis

- Díaz, G., y Salicetti, G. (2006). *Desarrollo de plataforma de procesamiento paralelo distribuido para la ejecución de diferentes trabajos*. Trabajo de Grado, Ingeniería en Informática, Universidad Católica Andrés Bello, Caracas.

- Licir, A. y Lucena W. (2006). *Sistema de detección de tipo y cantidad de vehículos presentes en una imagen digital*. Trabajo de Grado, Ingeniería en Informática, Universidad Católica Andrés Bello, Caracas.

- Becerra, A. y Loza, D. (2008). *Desarrollo e implementación de las estructuras y el ambiente de almacenamiento distribuido sobre el cluster de base de datos de una plataforma telemática de modelaje y simulación de tráfico vehicular*. Trabajo de Grado, Ingeniería en Informática, Universidad Católica Andrés Bello, Caracas.

GLOSARIO

A

Apache: Es un software (libre) servidor HTTP de código abierto para plataformas Unix (BSD, GNU/Linux, etc.), Windows, Macintosh y otras.

Application Programming Interface (API): 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.

Bus: Término usado para cualquier arreglo físico que provea la misma funcionalidad lógica que un bus eléctrico paralelo, es un conjunto de conductores eléctricos en forma de pistas metálicas impresas sobre la tarjeta madre del computador, por donde circulan las señales que corresponden a los datos binarios del lenguaje máquina con que opera el Microprocesador.

C

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.

Computación: Es la ciencia que se ocupa de las herramientas, tanto a nivel de hardware como de software, para la solución de los problemas en forma automática. Muchas veces es considerado sinónimo de informática, sin embargo la diferencia estriba en que la computación está íntimamente ligada a la tecnología de las computadoras mientras que la informática está más bien vinculada a las formas de uso de los resultados.

Conmutador (switch): Es un dispositivo electrónico de interconexión de redes de ordenadores. Un conmutador interconecta dos o más segmentos de red pasando datos de un segmento a otro. Los conmutadores se utilizan cuando se desea conectar múltiples redes, fusionándolas en una sola.

Canales de Procesamiento de Imágenes (CPI): Cada una de las distintas configuraciones acorde a las necesidades de las imágenes provenientes de distintos dispositivos de distinta ubicación en la aplicación DECA.

CPU: Central Processing Unit. Unidad central de procesamiento. Es el procesador que contiene los circuitos lógicos que realizan las instrucciones de la computadora.

D

Desviación Estandar: Es la raíz cuadrada de la varianza.

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.

H

Hardware: Todos los componentes físicos de la computadora y sus periféricos.

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.

Html: Siglas de Hyper Text Markup Language, Lenguaje de Marcas de Hipertexto, es el lenguaje de marcado predominante para la construcción de páginas web.

I

Interfaz: Conjunto de comandos y/o métodos que permiten la intercomunicación del programa con cualquier otro programa o entre partes (módulos) del propio programa o elemento interno o externo.

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.

L

LAN: Local Area Network o red de área local. Se trata de una red de comunicación de datos geográficamente limitada, por ejemplo, una empresa.

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".

Microprocesadores: Circuito integrado que contiene todos los elementos necesarios para conformar una CPU.

Multiprocesadores: Ordenador que cuenta con dos o más microprocesadores.

Multicomputadoras: Sistemas que se comunican enviando mensajes por buses muy cortos y rápidos, o por conmutador.

N

Nodo: Máquina perteneciente a un cluster.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.

O

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

P

Paralelización: Acto de paralelizar una aplicación, es decir, convertir en código para se procesado de forma paralela.

PHP: Lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas.

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

Promedio: En matemáticas, el promedio es un término para definir el valor característico de un conjunto de números, existen varios métodos para calcularlos como la media aritmética, media geométrica, media ponderada o la media armónica.

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

R

Rendimiento: Medida o cuantificación de la velocidad/resultado con que se realiza una tarea o proceso.

Red: (Network) en tecnología de la información, una red es un conjunto de dos o más computadoras interconectadas.

S

Segmentador: Clase de DECA que implementa una función de segmentación de imágenes según diversos criterios como color o tamaño.

Servidor: Computadora central de un sistema de red que provee servicios y programas a otras computadoras conectadas. Sistema que proporciona recursos (por ejemplo, servidores de archivos, servidores de nombres). En Internet este término se utiliza muy a menudo para designar a aquellos sistemas que proporcionan información a los usuarios de la red.

Sistema: Conjunto de partes que trabajan interrelacionadas para cumplir un objetivo común.

Software: Término general que designa los diversos tipos de programas usados en computación.

SSA: Diferencia entre filas. Suma de cuadrados del factor A.

SSB: Diferencia entre columnas. Suma de cuadrados del factor B.

SST: Número que representa una proporción de error en todas las mediciones. Suma de cuadrados de la interacción de todos los factores.

Suma de Cuadrados: Medida de variación en Probabilidad y Estadística.

Súper Cómputo: Utilización de computadoras con gran capacidad de cálculo, memoria, almacenamiento y comunicaciones (supercomputadoras).

T

Taxonomía de Flynn: Clasificación de arquitecturas de computadoras propuesta por Michael Flynn en 1972.

U

Usuario: Persona que utiliza un sistema.

V

Varianza: En teoría de probabilidad y estadística la varianza es un estimador de la dispersión de una variable aleatoria X de su media $E[X]$.

W

WAN: Wide Area Network o red de área amplia es una red generalmente construida con líneas en serie que se extiende a distancias mayores a un kilómetro.

Web: World Wide Web, Red Global Mundial, es un sistema de documentos de hipertexto y/o hipermedios enlazados y accesibles a través de internet.