ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

TESIS DE GRADO

"COMPUTACIÓN DISTRIBUIDA Y SU APLICACIÓN TECNOLÓGICA PARA INCREMENTAR LA EFICIENCIA DEL USO DE RECURSOS INFORMÁTICOS EN RED DE LA ESPOL"

Previa a la obtención del titulo de:

INGENIERO EN COMPUTACIÓN ESPECIALIZACIÓN SISTEMAS TECNOLÓGICOS

PRESENTADA POR:

GONZALO RAIMUNDO LUZARDO MOROCHO
LUIS FERNANDO VARGAS VILLAGÓMEZ

GUAYAQUIL - ECUADOR

2006

AGRADECIMIENTO

A todos quienes nos
ayudaron siendo
pilares importantes para la
presentación de este trabajo.
En especial a Katherine Chiluiza y
Xavier Ochoa quienes nos apoyaron
y alentaron en todo momento.

DEDICATORIA

A nuestros padres

TRIBUNAL DE GRADO

PRESIDENTE

Ing. Hernán Gutiérrez

DIRECTOR DE TESIS

Msc. Xavier Ochoa Chehab

MIEMBROS PRINCIPALES

Ing. Cristina Abad Robalino

Ing. Carlos Monsalve Arteaga

DECLARACIÓN EXPRESA

"La responsabilidad por los hechos, ideas y doctrinas expuestas en esta tesis, nos corresponden exclusivamente; y, el patrimonio intelectual de la misma, a la Escuela Superior Politécnica del Litoral"

(Reglamento de exámenes y títulos profesionales de la ESPOL)

Gonzalo Raimundo Luzardo Morocho

Luis Fernando Vargas Villagómez

RESUMEN

En el primer capítulo se describe las necesidades de procesamiento de grandes cantidades de información que tiene la ESPOL; y como la computación distribuida puede ayudarnos a suplirlas. De igual manera se establece los objetivos del trabajo de investigación. Además se analiza la forma en que la computación distribuida puede ofrecer nuevas posibilidades para el desarrollo de proyectos en una gran variedad de campos, no sólo dentro de la ESPOL si no a nivel nacional. Finalmente se analizará detalladamente tanto las ventajas como las desventajas de la computación distribuida sobre los sistemas centralizados actuales, para tener pleno conocimiento de los pros y los contras que involucran la implementación de esta clase de sistemas.

En el segundo capítulo se realiza un estudio acerca de la computación paralela, la cual es la base para los sistemas distribuidos y por ende para la computación distribuida. Además, se estudia los diferentes esquemas de clasificación de este tipo de sistemas, en particular la denominada taxonomía de Flynn. Posteriormente se abarca los diferentes modelos de programación en sistemas paralelos, y las consideraciones que se debe tomar al momento de tratar de resolver un problema utilizando esta metodología. Por último se toma mayor énfasis en lo que es la computación distribuida, analizando los diferentes modelos que se pueden utilizar para su implementación.

En el tercer capítulo se analizan las posibles soluciones, tanto de hardware como de software, para resolver problemas que requieren gran capacidad de cómputo. Entre las soluciones de hardware que se tratan son el uso supercomputadoras, computadoras de alto rendimiento y clusters de computadoras heterogéneas. Además se analizan las diferentes soluciones de middleware o software (PVM, MPI, BOINC) para la computación distribuida, poniendo énfasis en sus principales características, así como las ventajas y desventajas que estas poseen. Por último, y en base al análisis elaborado, se describe la solución que se ha seleccionado en base a los recursos informáticos que posee la ESPOL.

En el cuarto capítulo se explica todo lo referente al diseño del sistema distribuido de procesamiento, en base a un conjunto de requerimientos planteados, siendo uno de los principales el uso de recursos computacionales de los equipos conectados a la red de la ESPOL, e incluso conectados a Internet. El diseño del sistema fue creado considerando que el sistema se va a basar en la plataforma distribuida BOINC. En el transcurso de este capítulo se detalla de forma gráfica cada uno de los componentes que forman el sistema, describiendo sus características, funcionamiento, e interacción con los otros componentes del sistema.

En el quinto capítulo se hace una descripción detallada de los pasos seguidos para la instalación y configuración del sistema de procesamiento distribuido, en base al diseño propuesto en el capítulo anterior. Se describe la instalación y configuración de la plataforma distribuida y sus clientes distribuidos. Además se describe el análisis, diseño e implementación de una aplicación distribuida que es probada sobre el sistema.

En el sexto capítulo se realiza una descripción de las pruebas de funcionamiento usando la aplicación distribuida desarrollada. Se muestran resultados de las mediciones realizadas en el sistema: de rendimiento, escalabilidad, speedup (aceleración), entre otras; todo esto para poder hacer estimaciones y proyecciones del poder computacional. Además se muestran datos comparativos (en tiempo de procesamiento en la desencriptacion de una clave RSA) entre el sistema distribuido de procesamiento desarrollado, y sistemas tradicionales no distribuidos.

Finalmente se detallan las conclusiones del trabajo de investigación así como las recomendaciones para la popularización de esta tecnología de alto rendimiento para su uso en futuros proyectos de investigación.

ÍNDICE GENERAL

ÍNDICE DE GRÁFICOS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En la actualidad es muy común la existencia de proyectos de investigación que deben analizar una gran cantidad de datos; renderización de imágenes, simulación de fenómenos naturales, redes neuronales, son algunos ejemplos. El común denominador entre todos ellos es la gran cantidad de procesamiento que requieren. Para poder obtener toda esta cantidad, utilizan computadoras de gran capacidad de procesamiento o bien llamadas supercomputadoras, que por lo regular requieren de una gran inversión económica.

Sin embargo, existen otras alternativas mucho más accesibles y efectivas, que permiten utilizar el “poder computacional combinado” de computadoras disponibles conectadas en red.

Por otro lado, la ESPOL cuenta actualmente con una gran cantidad de computadoras distribuidas alrededor de todo el campus. La mayoría de estas conectadas en red y utilizadas en actividades que no requieren mucho trabajo de procesamiento. Por lo cual, resulta muy factible crear una computadora virtual que reúna la capacidad de procesamiento de todos éstas computadoras disponibles.

Estadísticamente se ha comprobado que el 80% del procesador de una computadora no se encuentra en uso cuando se realizan tareas de aplicaciones sencillas tales como procesadores de texto, hojas de cálculo, entre otras [REF.1], por lo que este porcentaje puede ser invertido en nuevas tareas controladas de forma remota, tareas que requieren de gran procesamiento y se pueden realizar por parte de muchas computadoras que formen parte de una red.

El presente trabajo describe el proyecto de implementación de una plataforma de computación distribuida que aprovecha los recursos informáticos de la ESPOL. Este proyecto tiene por objetivo crear una computadora virtual con gran capacidad de procesamiento, usando los ciclos ociosos de las computadoras en red disponibles en la universidad.

Como resultado de la implementación del proyecto se obtuvo una computadora virtual con un gran poder de procesamiento y con la característica de ser altamente escalable, la cual en la actualidad utiliza los ciclos ociosos de un grupo de computadoras de la ESPOL ubicadas en el Centro de Tecnologías de Información, y que cuenta además con la capacidad de usar los recursos de cualquier computadora conectada a la red de la universidad e incluso de Internet.

Para probar la potencialidad del sistema distribuido se desarrolló una aplicación distribuida que permita explotar sus capacidades de procesamiento. Dicha aplicación debía descifrar una clave RSA usando el método de fuerza bruta. Los resultados fueron prometedores, obteniendo datos muchos más rápidos comparados con sistemas de procesamiento no distribuidos.

CAPÍTULO 1

1 PLANTEAMIENTO DEL PROBLEMA Y ANÁLISIS CONTEXTUAL

En el presente capítulo se describe las necesidades de procesamiento de grandes cantidades de información que tiene la ESPOL; y como la computación distribuida puede ayudarnos a suplirlas. De igual manera se establece los objetivos del trabajo de investigación. Además se analiza la forma en que la computación distribuida puede ofrecer nuevas posibilidades para el desarrollo de proyectos en una gran variedad de campos, no sólo dentro de la ESPOL si no a nivel nacional. Finalmente se analizará detalladamente tanto las ventajas como las desventajas de la computación distribuida sobre los sistemas centralizados actuales, para tener pleno conocimiento de los pros y los contras que involucran la implementación de este tipo de sistemas.

1.1 Definición del problema

El desarrollo de la investigación abre nuevas necesidades de procesamiento en las diferentes unidades de la ESPOL, por ejemplo el campo de reconocimiento de imágenes, robótica, análisis de enfermedades del camarón, pronóstico del tiempo, biotecnología, minería de datos, entre otros.

Todas estas nuevas aplicaciones, necesitan de grandes capacidades de procesamiento, que son prohibitivas por la gran inversión que significa el proveerse de una supercomputadora. Con la implementación de la computación distribuida se puede obtener una computadora virtual formada por computadoras en red con una velocidad inclusive superior a la de una computadora central de última generación.

Estadísticamente se ha comprobado que el 80% del tiempo de un procesador en una computadora no se encuentra en uso cuando se realizan tareas de aplicaciones sencillas tales como procesadores de texto, hojas de cálculo, entre otras [REF.1], por lo que este porcentaje puede ser invertido en nuevas tareas controladas de forma remota, tareas que requieren de gran cantidad de

procesamiento y que pudiera realizarse distribuyéndolas en muchos computadores que formen parte de la red.

La ESPOL, cuenta actualmente con una gran cantidad de recursos informáticos que se han ido incrementando notablemente en los últimos años. Todos estos computadores se encuentran conectados en red y muchos de ellos no son usados de una manera eficiente por parte de los usuarios. Por lo que al existir la posibilidad de poder crear una máquina paralela virtual que reúna la capacidad de procesamiento de todos éstos, se podrá abrir camino en el campo de la investigación científica a niveles muy superiores de los actuales.

1.2 Objetivos del trabajo de investigación

El presente trabajo de investigación tiene por objetivo principal construir una supercomputadora virtual utilizando el poder de procesamiento combinado de las computadoras personales en la ESPOL.

De manera específica se establecen los siguientes resultados:

- Generar un sistema de computación distribuida en paralelo utilizando los ciclos ociosos de las computadoras personales de las áreas administrativas y de los laboratorios de la ESPOL.

- Proveer a la comunidad científica, investigadores, profesores y estudiantes del conocimiento y herramientas necesarias, que permitan explotar un sistema distribuido con altas capacidades de procesamiento para su uso en la investigación y el desarrollo.
- Elaborar una aplicación que explote las capacidades del supercomputador y demuestre sus posibles usos en la investigación.
- Sentar las bases para futuras investigaciones en el área de Procesamiento en Paralelo y Sistemas Distribuidos en la ESPOL.

1.3 Desarrollo de la computación distribuida

El constante avance tecnológico en el desarrollo de las computadoras ha sido gigantesco en estos últimos años, y a medida que se incrementa el poder computacional de ellas muchas computadoras con el pasar del tiempo se vuelven anticuadas y poco aprovechadas por gran parte de los usuarios; por otro lado tenemos que existen aplicaciones de investigación, en el área de desarrollo científico por ejemplo, que requieren de un gran poder computacional, por lo que se requiere del uso de computadoras ó supercomputadoras de última generación que generalmente tienen un alto costo.

De igual forma el desarrollo de las redes de computadoras de alta velocidad ha sido un paso muy importante para el desarrollo de las comunicaciones. Las redes de área local LAN dan la posibilidad de poder interconectar un sin número de computadoras, y de esta manera poder transmitir pequeños montos de información entre éstas en tiempos relativamente cortos. Las redes de área amplia WAN permiten a millones de computadoras alrededor del planeta conectarse entre sí a velocidades que varían de 64 Kbps a giga bits por segundo en algunas redes experimentales avanzadas.

Como resultado de esta realidad nace la idea de agrupar un número considerable de computadoras conectadas entre sí usando una red de alta velocidad, con el fin de poder aprovechar al máximo el uso de computadoras de diferentes arquitecturas que se encuentren conectadas en red, para realizar el procesamiento de datos en paralelo, y lograr de esta forma crear una supercomputadora virtual con el uso de los recursos que nos brindan estos equipos, capaz de procesar a una gran velocidad dichos datos.

La idea de hacer uso de todos aquellos ciclos ociosos del CPU, es tan antigua como las primeras redes que dieron paso al Internet [REF.2]. Las aplicaciones actuales que se han desarrollado a partir

de estas ideas han ido evolucionando con ellas, las cuales se basan en dos parámetros: qué terminales se encuentran disponibles para su uso y qué tipos de aplicaciones podrían ser ejecutadas.

Los diferentes tipos de proyectos de investigación que podemos encontrar hoy en día fueron desarrollados hace pocos años con el crecimiento de los usuarios de Internet y los accesos en redes de alta velocidad.

1.3.1 Historia

1.3.1.1 Los gusanos informáticos

Los primeros programas de computación distribuida aparecieron en los años 70, llamados Creeper y Reaper, los cuales se abrieron paso a través de los nodos de ARPANET[1]. El primero en aparecer fue Creeper al cual se lo denominó programa “gusano” el cual hacía uso de los ciclos ociosos del CPU en ARPANET para poder hacer una copia de sí mismo en un sistema próximo y luego borrarse del anterior [REF.2].

[1] ARPANET: Red Predecesora del Internet.

Después de su aparición, el Creeper fue modificado para mantener una copia de si mismo en todos los sistemas de la red. El Reaper fue creado con la intención de viajar a través de dicha red borrando las copias del Creeper del sistema. De esta manera, ambos programas se convirtieron en los primeros programas infecciosos. Actualmente se piensa que fueron los primeros virus que aparecieron en una red. Fue así que partir de estos programas se explotó la idea de aprovechar el poder computacional de los ciclos ociosos del CPU [REF.2].

Otro programa “gusano” el cual expandió la idea de los dos anteriores, fue el creado por John F. Shoch y Jon A. Hupp del Centro de Investigación de Xerox en Palo Alto en 1973. Dicho programa se movía de forma similar al Creeper y Reaper a través de cerca de 100 computadoras en una red local Ethernet usando los ciclos ociosos de cada CPU para realizar el rendering[1] de imágenes o gráficos por computadora [REF.2].

El primer proyecto de computación distribuida basado en Internet fue iniciado en 1988 por el Centro de Investigación de Sistemas DEC.

[1] El proceso de transformar datos matemáticos a una imagen o gráfico.

Este proyecto enviaba tareas a computadoras voluntarias por medio de correo electrónico, dichas computadoras ejecutaban estas tareas durante sus tiempos ociosos, para luego enviar los resultados hacia DEC y obtener una nueva tarea a realizar. La finalidad del proyecto era la de realizar factorización de números de gran tamaño, y para el año de 1990 contaba con alrededor de 100 usuarios [REF.2].

En los años 90, DEC y otros grupos de investigación, comenzaron a usar una interfase Internet para la distribución de tareas a ser realizadas; conducidos y auspiciados por proyectos propuestos por los Laboratorios de Investigación de RSA, RSA Security. Estos proyectos se caracterizaban por ser problemas de difícil solución, tales como factorización y búsqueda de números primos, y desencriptación. RSA auspició estos proyectos y herramientas de investigación, de esta forma ellos podrían resolver problemas que requerían de gran procesamiento computacional y realizar investigación en criptología. El problema del poder computacional fue la clave en todos estos proyectos, y los grupos de investigación unieron esfuerzos para cubrir más esta área [REF.2].

1.3.1.2 Beowulf

En 1994, el grupo de investigación CESDIS, liderado por Donald Becker y Thomas Sterling, con el auspicio del Centro Espacial de Vuelos Goddard de la NASA, en vista del gran costo que suponía el adquirir una supercomputadora, desarrollaron un sistema alternativo a éste. Consistía en un cluster[1] o grupo de ordenadores que interconectados entre sí, actuaban a velocidades de billones de operaciones por segundo (gigaflops) [REF.3].

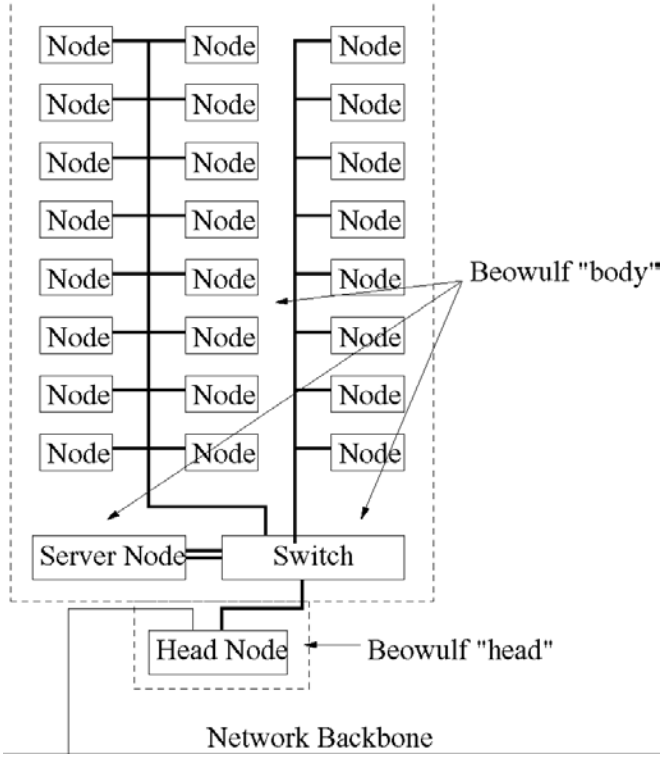


Figura 1.3.1.2-1. True Beowulf [F1]**.**

[1] Cluster: Un tipo de arquitectura que consiste en un grupo de ordenadores conectados entre sí a través de una red.

El "cluster" Beowulf original, formado por 16 procesadores Intel 486 DX4, conectados a través de una red Ethernet de 10 Mbps, fue el comienzo de una de las tecnologías competidoras de las supercomputadoras, gracias al bajo costo de los equipos y al uso de Linux como sistema operativo, hoy en día esta nueva forma de conseguir potencia a partir de pocos recursos cuenta con muchos seguidores en todo el mundo, gracias a lo cual ya hay un gran número de estas redes instaladas [REF.3].

1.3.1.3 Distributed.net

El grupo de investigación más destacado, siendo actualmente considerados como el primero en utilizar el Internet para distribuir datos para su cálculo y recopilación de resultados, fue el proyecto fundado en el año de 1997 llamado Distributed.net. Este grupo usó independientemente sus propias computadoras, tal como DEC lo había hecho; a diferencia que permitía al usuario poder descargar el programa que hiciere uso de los ciclos ociosos de su CPU, en lugar de enviar un mail para aquello. Distributed.net realizó varios proyectos en criptología propuestos por RSA Labs, así como también otros proyectos de investigación con la ayuda de miles de usuarios en línea [REF.4].

1.3.1.4 SETI@Home

El proyecto que realmente popularizó la computación distribuida, demostrando que en realidad podría funcionar, fue SETI@Home, un esfuerzo para la búsqueda de inteligencia extraterrestre (SETI[1]) desarrollado por la Universidad de California en Berkeley. El proyecto fue iniciado en Mayo de 1999 para analizar las señales de radio recopiladas por el Radio Telescopio Arecibo en Puerto Rico; dicho proyecto ha agrupado a más de 3 millones de usuarios quienes proveen de forma voluntaria los ciclos ociosos de sus computadores para buscar señales del espacio que podrían haber sido originadas por alguna inteligencia extraterrestre y no por la Tierra [REF.5].

1.3.2 Desarrollo de herramientas de software para el procesamiento distribuido

El desarrollo de librerías de paso de mensaje entre procesadores ha contribuido considerablemente en el desarrollo de los sistemas distribuidos, una característica a destacar de esta metodología es que no importa la arquitectura ni plataforma de software o hardware para procesar la información, como PVM, MPI, y BOINC que es una

[1] SETI: Búsqueda de Inteligencia Extraterrestre, por su siglas en inglés Search for Extraterrestrial Intelligence.

de las más importantes plataformas para el desarrollo de aplicaciones distribuidas hoy en día.

El desarrollo de **Parallel Virtual Machine (PVM)** dio inicio en el verano de 1989, cuando Vaidy Sunderan, profesor de la Universidad de Emory visitó el Laboratorio Nacional de Oak Ridge para realizar investigaciones con Al Geist en el área de la computación distribuida heterogénea. Ellos necesitaban de una plataforma para explorar esta nueva área y poder así desarrollar el concepto de Máquina Virtual Paralela (PVM). En el año de 1991, Bob Manchek, un investigador de la Universidad de Tenesse formó parte del grupo de investigación para desarrollar una versión portable y robusta de PVM (PVM 2.0) [REF.6].

El uso de PVM creció rápidamente alrededor del mundo, siendo la comunidad científica quien corría la voz de la utilidad de este software para la investigación. [REF.6].

La idea central del diseño de PVM fue la noción de una “Maquina Virtual”[1]. Un aspecto de la maquina virtual fue el cómo las tareas en

[1] Maquina Virtual: Un conjunto de computadoras heterogéneas conectadas en red, que para el usuario se muestra como una gran computadora paralela.

paralelo intercambiaban datos. En PVM esto fue llevado a cabo mediante el uso de paso de mensajes. La portabilidad fue considerada más importante que el rendimiento, por dos razones, la comunicación a través de Internet era lenta y; la investigación estaba enfocada hacia problemas con escalamiento, tolerancia a fallos y heterogeneidad de la maquina virtual [REF.6].

En Febrero de 1993, gracias a los numerosos aportes de los usuarios de PVM, se reescribió completamente el código de la herramienta, teniendo como resultado PVM 3. El software PVM ha sido distribuido gratuitamente, y es hoy en día uno de los más usados en el desarrollo de aplicaciones distribuidas alrededor del mundo [REF.7].

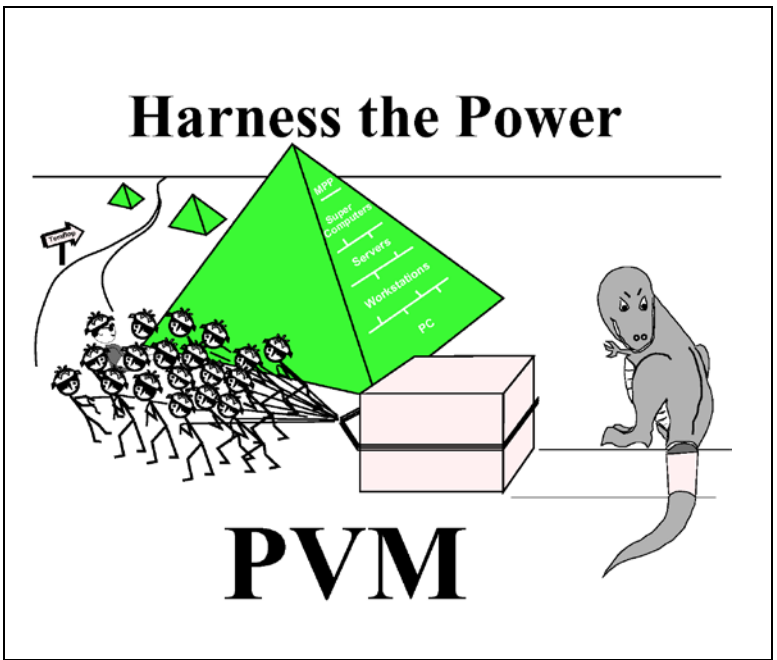


Figura 1.3.2-1. PVM [F2]

Por otro lado, **Message Passing Interface (MPI)** o llamado estándar MPI, cuyas especificaciones fueron culminadas en abril de 1994, es el sobrevenir de un esfuerzo comunitario para tratar de definir tanto la sintaxis como la semántica de un núcleo de rutinas de librerías de paso de mensajes que podrían ser muy útiles para los usuarios que desean implementarla de forma eficiente en un amplio rango de MPPs[1].

Uno de los objetivos del desarrollo de MPI es el de proveer a los fabricantes de MPPs un conjunto de instrucciones claramente definidas que puedan implementarse de forma eficiente, y en algunos casos, proveer el soporte para el hardware y por consiguiente el mejorar la escalabilidad de estos sistemas.

En cambio**, Berkeley Open Infrastructure for Network Computing (BOINC),** es un proyecto de computación distribuida desarrollado con la finalidad de ser usado en SETI@Home e ir más allá de los campos de investigación de este proyecto. BOINC fue desarrollado por un grupo de investigadores de la Universidad de California, Berkeley liderado por el director del proyecto SETI, David Anderson.

[1] MPPs: Sistemas de Procesadores Masivamente en Paralelo.

El objetivo de BOINC es la de dar la posibilidad a los investigadores de diversas áreas tales como, biología molecular, climatología, y astrofísica, de aprovechar el enorme pero subutilizado poder de cálculo de las computadoras personales alrededor del mundo. BOINC es una plataforma de software para computación distribuida que utiliza los recursos de computadoras voluntarias.

El proyecto SETI@home supuso un importante paso dentro del área de la computación distribuida. El concepto de utilizar el tiempo libre de los ordenadores conectados a Internet ha elevado la potencia de cálculo del proyecto a cimas absolutamente inalcanzables para cualquier computadora paralela convencional.

El PC puede ser usado, alternativamente, para buscar vida inteligente en el espacio, para solucionar el enigma del cáncer o realizar pronósticos meteorológicos. Todo ello, mientras el usuario del PC disfruta de su pausa de almuerzo o bebe un café. El sistema BOINC también hará posible poner a disposición de los proyectos, el espacio en el disco duro, además de la capacidad de cálculo del PC.

BOINC actualmente se encuentra en etapa de desarrollo y los únicos proyectos en funcionamiento para las pruebas son[1]:

- SETI@home: El proyecto de búsqueda de inteligencia extraterrestre analiza ondas de radio captadas con radiotelescopios en busca de un patrón inteligente.
- Predictor@home: El objetivo de este proyecto es el de encontrar soluciones a enfermedades relacionadas con proteínas mediante la predicción de las estructuras proteínicas a partir de sus secuencias.
- Climate Prediction: Es un proyecto que estudia modelos de cambio climático. Prediciendo lo que podría suceder con el clima dentro de 100 años.
- Astropulse: Es un proyecto que busca indicios de evaporación de agujeros negros. Es el proyecto con el que se han hecho las pruebas de BOINC

[1] Hasta un mes antes de la presentación de este documento estos eran los únicos proyectos para esta plataforma que se habían desarrollado.

1.4 Campos de acción y desarrollo de la computación distribuida

La computación distribuida ofrece nuevas posibilidades para el desarrollo de nuevos proyectos en una gran variedad de campos; como el científico, industrial y los negocios, entre otros; que antes se pensaron imposibles debido a las limitaciones de recursos, incluidos tiempo y dinero.

Gracias a la computación distribuida se pueden obtener resultados con una rapidez casi similar, y en muchas veces superior a la de una supercomputadora comercial, bajando el costo que implicaría desarrollar un sistema distribuido. A continuación veremos como la computación ayuda al desarrollo en los siguientes campos.

1.4.1 Campo de la investigación científica

Los proyectos científicos de comienzos de este siglo abordan objetivos cada vez más ambiciosos que requieren la resolución de problemas computacionales complejos, tanto por el volumen de los cálculos a realizar como por el tamaño y complejidad de las bases de datos utilizadas, SETI@home analiza una gran cantidad de señales de radio recopiladas por el Radio Telescopio Arecibo en Puerto Rico en busca de vida extraterrestre, por ejemplo. Del mismo modo, los

equipos científicos son en muchos casos colaboraciones internacionales con miembros distribuidos por todo el planeta. Áreas científicas tales como la Física de Altas Energías, Ciencias del Espacio, Genómica y Proteómica, o Meteorología, basan su desarrollo en estos proyectos.

Además en muchos casos, estos datos necesitan representar de forma gráfica y multidimensional el resultado de un análisis, requiriendo de una gran capacidad de procesamiento para observar los resultados finales.

La computación distribuida propone agregar y compartir recursos de computación distribuidos entre diferentes organizaciones e instituciones a través de redes de alta velocidad, de modo que el acceso a los mismos por parte de los científicos para sus necesidades de cálculo sea tan sencillo, flexible y fiable como el uso de la corriente eléctrica para satisfacer sus necesidades de energía. Muchas aplicaciones científicas hacen uso de esto, como por ejemplo el proyecto “Earth Simulator” desarrollado por la Corporación NEC, para la investigación y el pronóstico del tiempo [REF.8].

1.4.2 Campo de la industria

La industria de la animación 3D y las artes gráficas han sido una de las más beneficiadas gracias al creciente desarrollo de la computación distribuida. Hoy en día podemos disfrutar de películas y publicidad 3D que poseen un gran nivel de realismo, tales como Monster Inc., Final Fantasy y Shrek, entre otros. Para lograr este nivel de realismo se usan una gran cantidad de máquinas conectadas en paralelo que procesan la información para tener como resultado un corto de animación tridimensional. Hasta el momento no existe una computadora individual con el poder computacional necesario que sea capaz de producir una animación final de larga duración, debido a la gran cantidad de cálculos que se necesita hacer para lograr la gran cantidad de efectos especiales que incluye una producción de este tipo.

En la industria de las telecomunicaciones la computación distribuida puede mejorar considerablemente el procesamiento de tareas que llevarían inclusive días en poder realizarlas, tenemos por ejemplo el proceso de diseño y optimización de redes mediante la simulación de varias configuraciones posibles de red basadas en un sinnúmero de parámetros.

En la telefonía móvil hacia los servicios de tercera generación se puede implementar nuevos y mejores métodos de tarifación y detección de fraude con la aplicación de nuevos algoritmos que requieren gran cantidad de cálculo.

Existen industrias de elaboración de productos de consumo masivo, las cuales necesitan controlar y monitorear el procesamiento de varias de sus etapas de producción, aquí se recopilan miles de datos mediante varios dispositivos e interfaces, luego esta información debe ser procesada casi en tiempo real con el fin de obtener un análisis exacto de la producción en curso y de esta manera poder prevenir cualquier mal funcionamiento de procedimientos en la elaboración de sus productos.

1.4.3 Campo de los negocios

En el sector de los negocios, como lo es el de las finanzas, se necesita de una gran potencia de cálculo la cual afortunadamente es distribuible. La computación distribuida permite que tareas que normalmente requerirían horas e incluso días, como calcular el valor en riesgo usando el modelo de Monte Carlo [REF.10], optimizar carteras, cálculo de volatilidades implícitas para su uso en valoración de opciones y operaciones de cobertura, métodos de valoración de

opciones exóticas, puedan ser llevados a cabo en tan solo pocos minutos.

1.5 Ventajas de la computación distribuida

La computación distribuida es el procesamiento que se realiza sobre los sistemas distribuidos. Los sistemas de procesamiento distribuido nos ofrecen muchas ventajas, a continuación se describen algunas.

1.5.1 Ventajas sobre los sistemas centralizados

En general, la computación distribuida exhibe algunas ventajas sobre el procesamiento que se realiza en los sistemas centralizados, las cuales se describen continuación.

Razón precio/rendimiento

En computadoras normales, no se obtiene el doble del rendimiento por el doble del precio. La curva Precio/Rendimiento tiende a ser no lineal y pronunciada. Con múltiples CPUs, podemos tener casi el doble del rendimiento por el doble de precio. Una de las razones principales para implementar un sistema de procesamiento distribuido es que la razón Precio/Rendimiento es mucho mejor en

estos sistemas si la comparamos con la de un sistema centralizado [REF.9].

Velocidad de procesamiento

Un sistema de procesamiento distribuido puede en algún momento tener mucho más poder computacional que el de un sistema de procesamiento centralizado [REF.9].

Alta confiabilidad

Al distribuir el trabajo sobre muchas máquinas hace que un sistema de procesamiento distribuido sea muy confiable, ya que la falla de un ordenador no afecta al sistema, a tal punto que puede seguir operando si ningún problema [REF.9].

Desarrollo incremental

Una compañía para satisfacer las necesidades de procesamiento puede comprar un ordenador, sin embargo en algún momento la carga de trabajo será demasiada para dicho ordenador. La única opción es ese momento sería comprar un nuevo ordenador que se adapte a las nuevas necesidades de procesamiento, si es que existe. En un sistema de procesamiento distribuido el poder computacional puede ser mejorado con sólo agregar más procesadores.

Economía	La razón Precio/Rendimiento es mucho mejor en el sistema de procesamiento distribuido si la comparamos con la de un sistema centralizado.
Velocidad de Procesamiento	Un sistema de procesamiento distribuido puede en algún momento tener mucho más poder computacional que un sistema centralizado.
Confiabilidad	La falla de un ordenador no afecta al todo el sistema.
Desarrollo incremental	En un sistema de procesamiento distribuido el poder computacional puede ser mejorado con sólo agregar más procesadores.

Tabla 1.5.1-1. Ventajas de la computación distribuida sobre los sistemas de procesamiento centralizados [REF.11].

1.5.2 Ventajas sobre las computadoras personales

Un sistema de procesamiento distribuido es potencialmente más flexible. A pesar de que un modelo de computación distribuida sería darle a cada persona un computador individual, todos estos conectados entre si mediante una LAN, no es la única posibilidad. Otra sería tener una mezcla de computadoras personales y compartidas, quizás de diferente tamaño, y asignar trabajo apropiado a cada una de ellas.

1.6 Desventajas de la computación distribuida

A pesar de que los sistemas de procesamiento distribuido tienen muchas ventajas, también presentan desventajas. En esta parte

veremos las desventajas de los sistemas de procesamiento distribuido.

Software

Con el actual estado de arte, no se tiene mucha experiencia en el diseño e implementación de aplicaciones para el procesamiento distribuido. ¿Qué clase de sistemas operativos, lenguajes de programación y aplicaciones son apropiadas para el desarrollo de este tipo de sistemas?, son algunas de las preguntas que nos hacemos. Hoy en día, cuanto más avanza la investigación en este campo, este problema se va disminuyendo, pero por el momento no se lo puede desestimar [REF.9].

Red

Otro problema con respecto a los sistemas de procesamiento distribuidos es ocasionado por las redes de comunicación. La red puede perder los mensajes que se envían entre las computadoras, lo que ocasiona que la red se sobrecargue. Una vez que el sistema depende de la red, en pérdida de datos o saturación de la red, puede hacer que la ventaja de usar un sistema de procesamiento distribuido no se aprecie. Además, mejorar la red podría ser muy costoso y en muchos casos muy difícil [REF.9].

Seguridad

El tener computadoras conectadas entre sí a través de una red implica un problema de seguridad. Una computadora podría acceder de manera sencilla a datos de otra computadora en la misma red.

Software	Actualmente no se tiene mucha experiencia en el diseño e implementación de aplicaciones para el procesamiento distribuido
Red	La red puede saturarse o causar otros problemas.
Seguridad	Fácil acceso implica problemas de seguridad.

Tabla 1.6-1. Desventajas de los sistemas distribuidos [REF.12].

1.7 Futuro de la computación distribuida

La computación distribuida está alcanzando la madurez, mucho más si hoy en día está siendo combinada con servicios Web para crear una Internet superpotente, según lo expresado por los participantes del GGF[1].

[1] Global Grid Forum: Foro de miles de usuarios interesados en la normalización global de la programación en red.

Los últimos foros han estado centrados en la integración de la computación distribuida y los servicios Web: la red brinda el servicio, y el Grid[1] aporta la potencia de procesamiento necesaria para ello. Para esto el GGF desarrolló OGSA[2], equivalente a TCP/IP, que brinda dicha integración.

Hay muchas áreas en donde se verán usos masivos y espectaculares de esta tecnología. Algunos ya están en marcha como SETI@home, que utiliza millones de máquinas dispersas para procesar las señales que provienen del espacio, fuera del sistema solar, buscando detectar vida extraterrestre, o Jabber, una herramienta que permite mediante el uso de agentes inteligentes diseñar en forma automática comunidades virtuales.

Los juegos es otra área donde la integración de la computación distribuida y los servicios Web darán lugar a una nueva y superpotente Internet. Sony y Microsoft están poniendo su mirada en

---

[1] Conjunto de computadores conectados entre si, mediante Internet, por ejemplo. A la computación distribuida también se la denomina Grid Computing porque usa recursos de ordenadores conectados entre sí.

[2] Open Grid Services Architecture, un conjunto de especificaciones y estándares que combina los beneficios de la informática Distribuida sobre Internet y los servicios Web.

los juegos distribuidos, mientras que Butterfly.net es otro de los proyectos en los que se está desarrollando juegos multijugador masivos basados en esta tecnología.

No obstante, la gran ciencia seguirá siendo una parte fundamental del futuro de la computación distribuida. Un claro ejemplo de ello es la TeraGrid[1] que reúne todas las condiciones para robarle el segundo puesto al ASCI White[2] en el ranking mundial de las supercomputadoras.

Según se espera cuando esté lista, la TeraGrid tendrá 20 teraflops de potencia computacional Linux Cluster distribuidos en los cinco sitios de la TeraGrid, lo cual le permitirá procesar y almacenar cerca de 1 petabyte[3] de información. Estará conectada a través de una red de 40 Gbps, con lo cual se transformará en una red de 50 a 80 Gbps, es decir, 16 veces más veloz que la red de investigación más rápida que

---

[1] Proyecto para la construcción y desarrollo de la infraestructura distribuida para la investigación científica más grande y rápida del mundo.

[2] Por sus siglas en inglés Accelerated Strategic Computing Inicitiave, supercomputadora creada por la IBM.

[3] Un petabyte corresponde a 2^{50} bytes

existe actualmente. La misma será utilizada para aplicaciones comerciales y proyectos auspiciados por la NSF[1].

Sin duda alguna el futuro de la computación distribuida es prometedor. Muchas aplicaciones y proyectos distribuidos llegarán a ser el pan de cada día. Quizás algún día, las compañías, agencias del gobierno, y centros de investigación podrían incluso pagar para poder usar los ciclos ociosos de las computadoras, y contribuir en la búsqueda de soluciones a grandes problemas, encontrando quizás algún día una cura para el cáncer, por ejemplo.

[1] Fundación Nacional de Ciencias por sus siglas en inglés: National Science Foundation.

CAPÍTULO 2

2 COMPUTACIÓN PARALELA Y SISTEMAS DISTRIBUIDOS

En el presente capítulo se realiza un estudio acerca de la computación paralela, la cual es la base para los sistemas distribuidos y por ende para la computación distribuida. Además, se estudia los diferentes esquemas de clasificación de este tipo de sistemas, en particular la denominada taxonomía de Flynn.

Posteriormente se abarca los diferentes modelos de programación en sistemas paralelos, y las consideraciones que se deben tomar al momento de tratar de resolver un problema utilizando esta metodología.

Por último se toma mayor énfasis en lo que es la computación distribuida, analizando los diferentes modelos que se pueden utilizar para su implementación.

2.1 Introducción

La computación paralela o procesamiento paralelo, método por el cual se puede dividir un problema grande en varias tareas para que su solución sea rápida de hallar, ha surgido como una tecnología prometedora de procesamiento en computación moderna. En los últimos años se ha incrementado la aceptación y adopción del procesamiento paralelo tanto para computación científica de alto rendimiento, como para aplicaciones de propósito general, todo esto como resultado de la demanda de alto rendimiento, bajo costo y productividad sostenida. Esta aceptación ha sido posible gracias a los dos desarrollos más importantes en computación paralela: *Procesadores Masivamente Paralelos (MPP)* y la popularización del uso de *Computación Distribuida*.

Los **MPPs** hoy en día son las computadoras más potentes en el mundo. Estas máquinas pueden combinar el poder computacional de unos cuantos miles de CPUs en un solo gran gabinete conectado a miles de gigabytes de memoria. Estas máquinas ofrecen un enorme poder computacional y son usadas para resolver “Problemas de Gran Desafío” como por ejemplo el modelado del clima global o el desarrollo de alguna medicina para curar algún tipo de enfermedad. Mientras las simulaciones se vuelvan más realísticas, el poder

computacional requerido para producirlas crece. De esta forma, los científicos dirigen sus investigaciones en MPPs y procesamiento paralelo a fin de obtener el mayor poder computacional que sea posible.

El segundo principal desarrollo que afectó la manera de resolver problemas de índole científico es la **computación distribuida**. La computación distribuida es el proceso por el cual un grupo de computadoras conectadas a través de una red son usadas colectivamente para buscar una solución a un gran problema. Hoy en día muchas organizaciones ya poseen redes de área local de alta velocidad, interconectando una gran cantidad de estaciones de trabajo, el poder computacional de estos recursos combinados puede fácilmente llegar a exceder el poder de una computadora de alto rendimiento. En algunos casos varios MPPs han sido combinados usando computación distribuida para producir un poder de computación inigualable.

El factor más importante en la computación distribuida es el costo. Grandes sistemas MPPs pueden llegar a costar más de 10 millones de dólares. Por otro lado, los usuarios notan un costo mucho menor

al tratar de resolver algún problema usando un conjunto de máquinas locales ya existentes.

Algo en común entre la computación distribuida y los MPP es la noción de paso de mensajes. En el procesamiento en paralelo, las tareas deben intercambiar datos entre sí. Un sinnúmero de paradigmas han sido creados para lograr este propósito, como la memoria compartida, compiladores paralelos y el paso de mensajes.

2.2 La computación paralela

2.2.1 Definición de computación paralela

El paralelismo se encuentra implícito en la mayoría de los procesadores modernos (ILP) y los sistemas operativos (Multiprocesamiento asimétrico y simétrico), sin embargo a pesar de que todo computador tenga algún grado de paralelismo, no podemos llamarlo computador paralelo debido a que la concurrencia de estos está oculta totalmente al programador [REF.13].

Procesamiento en Paralelo es el procesamiento de información que enfatiza el manejo concurrente de conjuntos de datos entre varios procesadores con el objetivo de resolver un solo problema [REF.13].

Computadora Paralela es una computadora formada por muchos procesadores, la cual es capaz de realizar procesamiento paralelo [REF.13].

De manera más sencilla, computación paralela es el uso simultáneo de múltiples recursos informáticos, con el fin de resolver un problema computacional [REF.14].

Los **recursos informáticos** pueden consistir en:

- Una única computadora con múltiples procesadores;
- Un número arbitrario de computadoras conectadas por medio de una red.
- Una combinación de ambos.

Un **problema computacional** usualmente demuestra características como la habilidad de:

- Poderse dividir en piezas discretas de trabajo que puedan ser resueltas simultáneamente.
- Ser ejecutado en varias instrucciones en cualquier momento.
- Resolverse en menos tiempo usando múltiples recursos computacionales que con uno solo.

Una **supercomputadora** es una computadora de propósito general capaz de resolver problemas individuales con desempeños muy altos respecto de otros construidos en la misma época; todas las supercomputadoras son paralelas, algunas formadas por pocos procesadores que tienen la característica de ser rápidos y costosos; y otras formadas por muchos procesadores baratos.

En la supercomputación básicamente se diferencian dos tipos de arquitecturas: las vectoriales y las distribuidas [REF.15].

- Los sistemas vectoriales suelen ser mucho más caros por el hecho de estar construidos con una tecnología mucho más vanguardista. Las prestaciones de estos sistemas pueden llegar a ser incomparables con las modestas computadoras domésticas del mercado de gran consumo, aún contando con solamente cuatro procesadores, por ejemplo. En estos sistemas se apuesta por un nivel tecnológico superior, con todo lo que esto conlleva. Un ejemplo de un sistema vectorial es un MPP.

- Los sistemas distribuidos o clusters, como su propio nombre indica, implementan una política de reparto y descentralización de recursos. Entre las ventajas con las que cuentan básicamente sobresalen una mayor escalabilidad y un menor costo de

construcción. Ambos factores han sido decisivos para atraer a un público más extenso, con menor poder adquisitivo aunque no por esto menos exigente.

2.2.2 ¿Por qué usar computación paralela?

Existen dos razones principales para usar computación paralela:

- Ahorrar tiempo.
- Resolver problemas que requieran de una gran potencia de cálculo, como por ejemplo en la Fig. 2.2.2-1 se muestran algunas aplicaciones que requieren de gran capacidad de procesamiento junto con el tiempo estimado de ejecución.

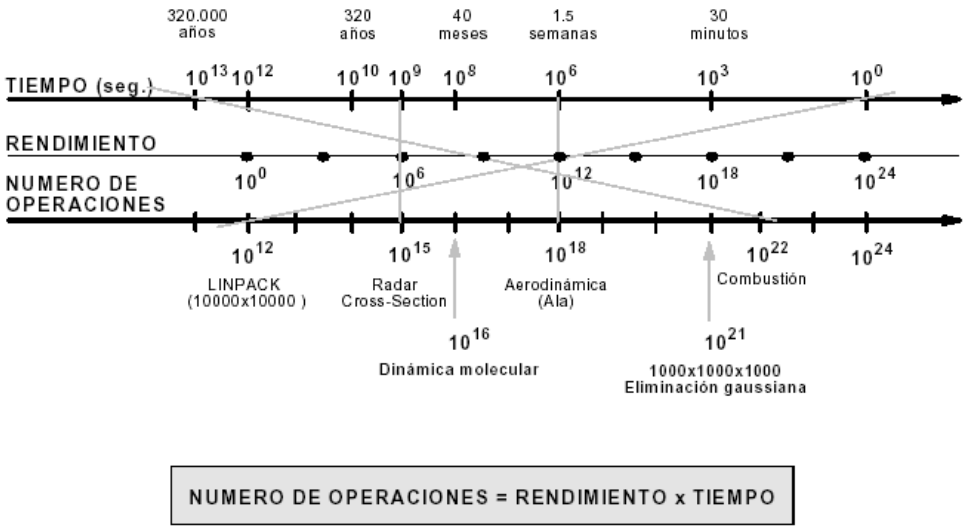


Figura 2.2.2-1. Problemas necesitados de gran potencia de cálculo [F3].

Otras razones podrían ser:

- **Aprovechar recursos no locales:** usar los que puedan estar disponibles en otras redes e incluso Internet, cuando los recursos locales son pocos o muy escasos.
- **Reducir costos:** usar recursos computacionales baratos, en lugar de pagar por el tiempo de uso de una supercomputadora.
- **Superar límites de memoria:** una computadora de escritorio tiene recursos de memoria muy limitados. Para grandes problemas, el uso de las memorias de varias computadoras puede superar este obstáculo.

2.2.3 Definiciones y terminologías

2.2.3.1 Rendimiento de trabajo (Throughput)

Uno de los factores importantes a considerar en el procesamiento paralelo es el rendimiento de trabajo (Throughput), el cual relaciona la cantidad de resultados obtenidos por unidad de tiempo. El throughput se puede mejorar de las siguientes maneras:

- Aumentando la velocidad del dispositivo (procesador).
- Aumentando la concurrencia, es decir, el número de operaciones simultáneas. La concurrencia se puede aumentar a través de la segmentación (pipelining) y el paralelismo de datos.

2.2.3.2 Segmentación (Pipelining)

El pipelining es una técnica que permite aumentar el grado de concurrencia de un procesamiento, actualmente esta técnica es utilizada intensivamente en el ILP[1].

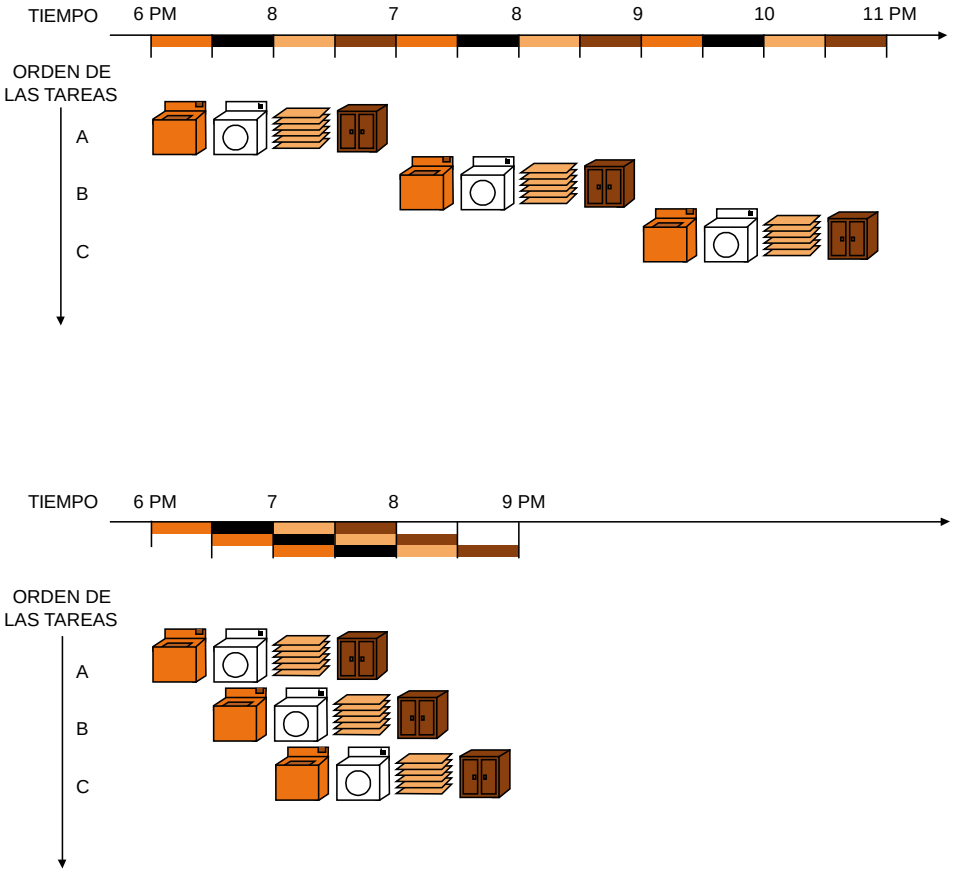


Figura 2.2.3.2-1. Solución secuencial vs. pipelined

Una computación Pipelined se divide en un número de pasos, llamados Pipe Stages o Segmentos (Segmentación), cada uno

[1] Paralelismo a nivel de instrucción.

trabajando a su máxima velocidad en cada etapa. La salida de un segmento corresponde a la entrada de otro segmento. Si todos los segmentos trabajan a la misma velocidad, el throughput del pipeline completo, es igual al throughput de un segmento cuando el pipe está lleno.

De manera más sencilla podemos decir que el procesador es organizado de manera similar a una línea de producción en una fábrica; varias instrucciones pueden estar en diferentes estados de ejecución de manera simultánea. Este solapamiento, aunque requiere hardware mucho más avanzado, nos garantiza una reducción significativa del tiempo total de ejecución de una secuencia de instrucciones [REF.16].

En la Fig. 2.2.3.2-1 tenemos un ejemplo de pipeline en el proceso de lavar, secar, planchar y guardar una cierta cantidad de ropa; mientras que de manera secuencial se debe esperar que finalice la secuencia completa; usando pipeline podemos tener varias secuencias que ejecuten varias instrucciones de manera simultánea, lo que ocasiona que se reduzca de manera significativa el tiempo total de procesamiento.

2.2.3.3 Paralelismo de datos

Es el uso de múltiples unidades funcionales con el fin de poder aplicar la misma operación de forma simultánea a varios elementos de un conjunto de datos, de esta forma un incremento de k veces en el número de unidades funcionales, lleva a incrementar en k veces el throughput de un sistema distribuido cuando no hay overhead asociada con el aumento del paralelismo.

De manera más simple podemos decir que paralelismo de datos significa la ejecución de la misma operación, sobre diferentes partes de un conjunto de datos normalmente grande [REF.17].

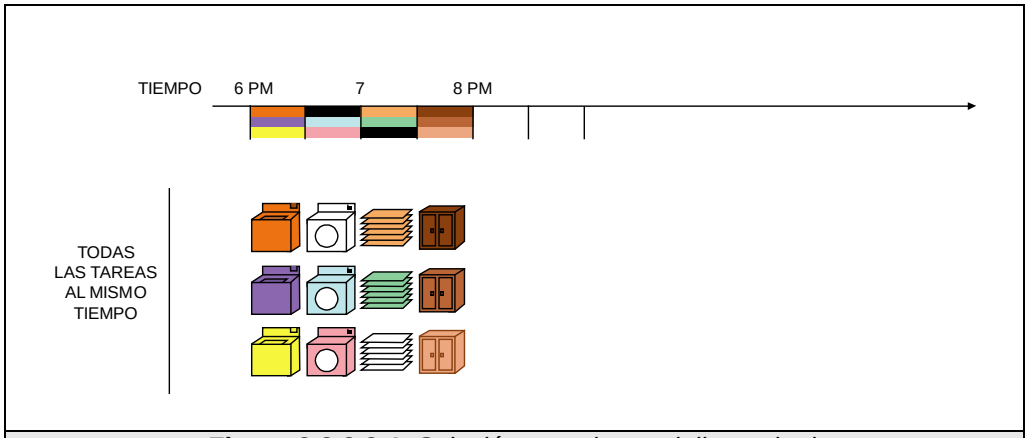


Figura 2.2.3.3-1. Solución usando paralelismo de datos

Siguiendo con el ejemplo anterior, en la Fig. 2.2.3.3-1 vemos la solución usando paralelismo. Cada máquina ejecuta todas las tareas. El throughput aumenta replicando las unidades funcionales.

2.2.3.4 Aceleración (Speedup)

Un término asociado con el paralelismo es el Speedup *(Sp)* o Aceleración, el cual permite relacionar los tiempos de ejecución de algoritmos secuenciales vs. Paralelos. El Speedup es una medida que nos indica qué tan rápido un programa determinado corre cuando se ejecuta sobre varios procesadores en paralelo, comparado con una ejecución de manera serial sobre un simple procesador [REF.18]. El Speedup se lo puede definir como:

$$Sp = \frac{Tiempo\_de\_ejecución\_del\_algoritmo\_más\_eficiente}{Tiempo\_de\_ejecución\_del\_algoritmo\_en\_una\_máquina\_que\_incorpora\_paralelismo}$$

2.2.3.5 Paralelismo de control

La técnica de Pipelining es tan solo un caso especial de una clase más general de algoritmos paralelos llamada paralelismo de Control.

Al contrario del paralelismo de datos, en el cual se aplica una misma operación a un conjunto de datos, el paralelismo de Control es un paralelismo que se logra mediante la aplicación simultánea de diferentes operaciones a diferentes elementos de datos [REF.19].

El flujo de datos entre los procesos puede ser arbitrariamente complejo, cuando el grafo de flujo de datos forma un grafo dirigido simple, se dice que el algoritmo es pipelined. Sin embargo, se tiene

que la mayoría de problemas reales exhiben los dos tipos de paralelismo, teniendo además una relación de precedencia entre las diferentes tareas.

2.2.3.6 Escalabilidad

Se dice que todo algoritmo es escalable si aumenta el nivel de paralelismo al menos linealmente con el tamaño del problema.

Una arquitectura es escalable, si al aumentar el tamaño del problema, el desempeño por procesador se mantiene aunque se aumente el número de procesadores.

En comparación, el paralelismo de datos posee mejor escalabilidad que el paralelismo de control, porque el nivel de paralelismo en paralelismo de control es usualmente una constante que es independiente del tamaño del problema.

2.2.3.7 Tareas simples y tareas paralelas

Una **tarea simple** es una sección lógica y discreta de trabajo computacional. Una tarea comúnmente es un programa visto como un conjunto de instrucciones que son ejecutadas por un procesador.

Una **tarea paralela**, en cambio, puede ser ejecutada en varios procesadores de manera segura, esto es, producir resultados correctos.

2.2.3.8 Comunicación y sincronización

Por lo regular las tareas paralelas necesitan realizar intercambio de datos entre sí. Existen diferentes maneras hacerlo; usando memoria compartida o a través de la red; sin embargo el evento real de intercambio de información es a menudo llamado comunicación, cualquiera que sea el método empleado.

La coordinación de tareas paralelas en tiempo real, muy a menudo es asociada con comunicaciones. Por lo regular esta coordinación es implementada estableciendo un punto de sincronización dentro de la aplicación; donde una tarea no puede continuar mientras que otras no alcancen el mismo punto.

La sincronización normalmente involucra esperar por lo menos por una tarea, y puede causar que el tiempo de ejecución de las aplicaciones paralelas incremente.

2.2.3.9 Granularidad

En computación paralela, granularidad se refiere al número de pequeñas secciones que un problema es dividido [REF.20].

- Gruesa: el problema es dividido en tareas relativamente grandes.
- Fina: el problema es dividido en tareas relativamente pequeñas.

2.2.3.10 Velocidad de ejecución

La velocidad de ejecución *(V)*, mide el número de salidas que se obtienen por unidad de tiempo. Según la naturaleza de las salidas tenemos:

- $V = \frac{Numero\_de\_instrucciones}{segundo}$, que se miden en MIPS (Millones de instrucciones por segundo).
- $V = \frac{Numero\_de\_operaciones}{segundo}$, que se miden en MOPS (Millones de operaciones por segundo).
- $V = \frac{Numero\_de\_operaciones\_de\_punto\_flotante}{segundo}$, que se miden en MFLOPS (Millones de operaciones de punto flotante por segundo).

2.2.3.11 Eficiencia

La eficiencia *(E)*, es la relación entre la aceleración de una ejecución paralela y el número de procesadores *(p)* [REF.21].

$$E = \frac{Sp}{p}$$

2.2.3.12 Redundancia

La redundancia *(R),* es la relación entre el número de operaciones realizadas utilizando *p* procesadores en una ejecución paralela y su correspondiente ejecución serie en 1 procesador [REF.21].

$$R = \frac{Op}{O1}$$

2.2.3.13 Utilización

La utilización *(U),* es la relación entre Op y el número de operaciones que podrían realizarse utilizando *p* procesadores en *Tp* unidades de tiempo [REF.21].

$$U = \frac{Op}{pTp}$$

2.2.3.14 Ley de Amdhal

La ley de Amdhal, desarrollada por Gene Amdhal en el año de 1967 la cual introdujo el concepto del Speedup en el mundo, es una forma

de expresar el máximo Speedup en función del paralelismo y de la fracción secuencial dentro de una tarea.

El Speedup en general nos permitía medir el incremento logrado al realizar una mejora para obtener un mejor desempeño:

$$Speedup = \frac{Tiempo\_de\_ejecucion\_de\_la\_tarea\_completa\_\text{sin}\_usar\_mejora}{Tiempo\_de\_ejecucion\_de\_la\_tarea\_completa\_usando\_la\_mejora\_cuando\_sea\_posible}$$

La fórmula de Amdhal se deriva de la aplicación de las siguientes fórmulas:

$$Speedup = \frac{Tviejo}{Tnuevo}$$

$$Fc/m = Fracción\_donde\_se\_usa\_la\_mejora$$

$$Fs/m = Fracción\_donde\_no\_se\_usa\_la\_mejora$$

$$Tnuevo = Tviejo \times Fs/m + Tusando\_solo\_la\_mejora \times Fc/m$$

$$Speedup\_local = \frac{Tviejo}{Tusando\_solo\_mejora}$$

$$Tnuevo = Tviejo \times (1 - Fc/m) + \frac{Tviejo \times Fc/m}{Speedup\_local}$$

$$Speedup = \frac{1}{(1 - Fc/m) + \frac{Fc/m}{Speedup\_local}}$$

La ley de Amdhal llevada al paralelismo produce consecuencias dramáticas, pues considerando que *f* es la fracción netamente secuencial de un programa y *p* el número de procesadores; el máximo Speedup que se puede lograr está dado por:

$$Sp \leq \frac{1}{f + \frac{1-f}{p}}$$

La ley de Amdhal expresa la ley de las ganancias que disminuyen, debido a que si una mejora usa sólo una fracción de tiempo que dura la tarea, no se puede apurar la tarea más que 1/f.

$$\lim Speedup_{(global)} = \frac{1}{f}$$

$$Speedup_{(local)} \rightarrow \infty$$

2.2.4 Interconexión de sistemas paralelos

2.2.4.1 Taxonomía de Flynn

Con el paso de los años se han desarrollado muchas propuestas en referencia a los diferentes esquemas para clasificar los sistemas de computadoras ya sean estos secuenciales o paralelos.

El modelo ampliamente aceptado es el propuesto por Flynn en 1972, denominado Taxonomía de Flynn [REF.22]. Para su definición Flynn se basó en dos características que consideró importantes:

- *El número de flujo de instrucciones*, que se refiere al número de secuencia de instrucciones ejecutadas por una computadora.
- *El número de flujo de datos*, que se refiere a la secuencia de datos manejados por una computadora.

De esta taxonomía se hace la caracterización de una arquitectura en base a la multiplicidad del hardware usado para manejar las instrucciones, y datos que debe manejar una computadora.

La multiplicidad se toma como el número máximo posible de instrucciones simultáneas o datos que están en la misma fase de ejecución en el componente con mayores restricciones de la máquina, dando como resultado la siguiente clasificación:

- SISD (Single instruction, single data).
- SIMD (Single instruction, multiple data).
- MISD (Multiple instructions, single data).
- MIMD (Multiple instruction, multiple data).

2.2.4.2 SISD

SISD (*Single Instruction, Single Data*), es en realidad un sistema con un solo procesador en el cual existe un flujo único de instrucciones y de datos, como se puede ver en la Fig. 2.2.4.2-1 [REF.23].

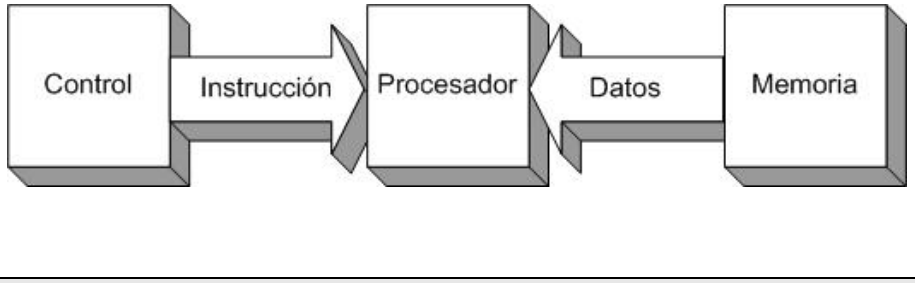


Figura 2.2.4.2-1. Modelo SISD

En la actualidad algunos servidores poseen más de un procesador, pero cada uno ejecuta instrucciones que no están relacionadas entre sí; por lo tanto estos sistemas se los considera también máquinas SISD. Ejemplos de este tipo de máquinas son la mayoría de computadoras personales o estaciones de trabajo.

2.2.4.3 SIMD

SIMD (*Single Instruction, Multiple Data*), es un sistema en el cual la misma instrucción es ejecutada de manera sincrónica por todos los procesadores, como se puede ver en la Fig. 2.2.4.3-1 [REF.23].

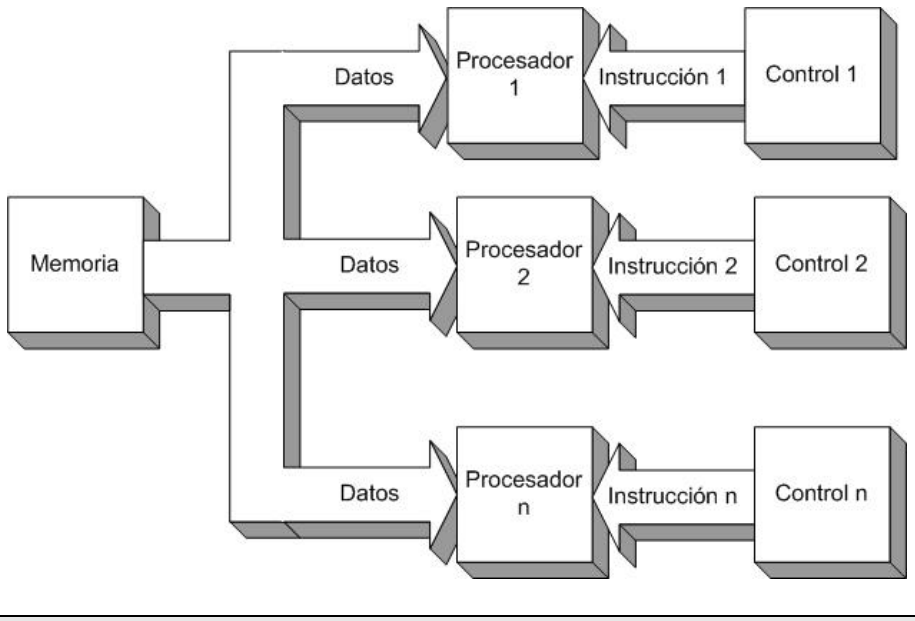


Figura 2.2.4.3-1. Modelo SIMD

Cada unidad de procesamiento puede operar sobre un elemento diferente de dato. Este tipo de máquinas usualmente trabajan como despachadores de instrucciones; una red de alta velocidad comunica a un gran número de unidades de procesamiento que por lo regular son de poca capacidad. Son diseñadas especialmente para resolver un tipo de problema en particular, como por ejemplo un procesamiento de imagen.

Un ejemplo de este tipo de sistemas es el SETI@home, ya que existe una gran cantidad de computadoras (unidades de procesamiento)

conectados a través de Internet; todos ellos tratando de resolver un problema; que en este caso es encontrar vida extraterrestre.

2.2.4.4 MISD

MISD (*Multiple Instruction, Single Data*), es un sistema en el cual se segmenta el programa y se lo divide entre múltiples procesadores, de tal manera que cada uno realiza una instrucción diferente, como se puede ver en la Fig. 2.2.4.4-1 [REF.23].

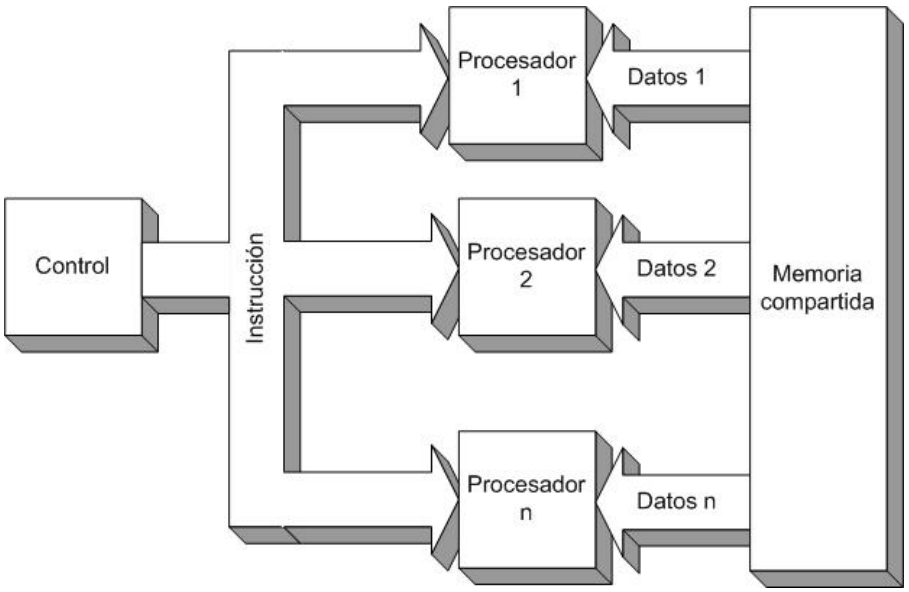


Figura 2.2.4.4-1. Modelo SIMD

Este tipo de máquinas son solamente teóricas, ya que no se ha construido ningún sistema práctico que pertenezca a esta clase.

2.2.4.5 MIMD

MIMD (*Multiple Instruction, Multiple Data*), es un sistema en el cual se ejecuta un programa diferente en cada procesador, como se puede ver en la Fig. 2.2.4.5-1 [REF.23].

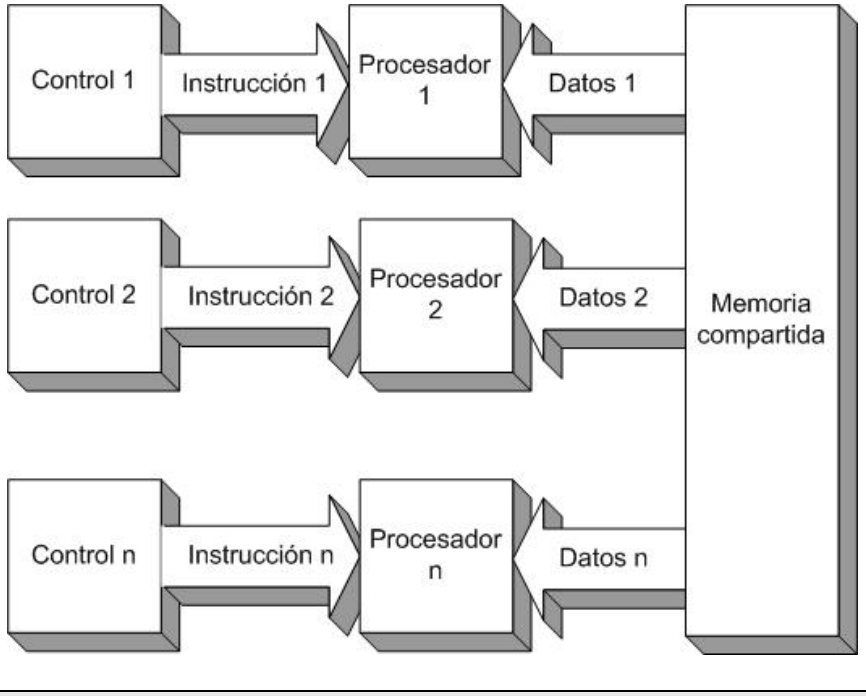


Figura 2.2.4.5-1. Modelo MIMD

La diferencia con los sistemas SISD de múltiples procesadores es que las instrucciones y los datos están relacionados, ya que representan diferentes partes de una misma tarea. Los **sistemas distribuidos**, una clase de sistemas paralelos, pertenecen a esta clase.

Existen gran variedad de sistemas MIMD, tales como el nCUBE 3, el Cray Y-MP T94, entre otros.

Aunque la Taxonomía de Flynn se detiene en los MIMD; en la Fig. 2.2.4.5-2 podemos ver que éstos a su vez se pueden dividir en dos grupos [REF.24]:

- Multiprocesadores, aquellos que tienen memoria compartida. Un espacio virtual de memoria es compartido por todos los CPUs.
- Multicomputadoras, aquellos que no poseen memoria compartida. Donde cada máquina tiene su propio espacio de memoria. Un ejemplo común de multicomputadoras es un grupo de computadoras personales conectados a través de una red.

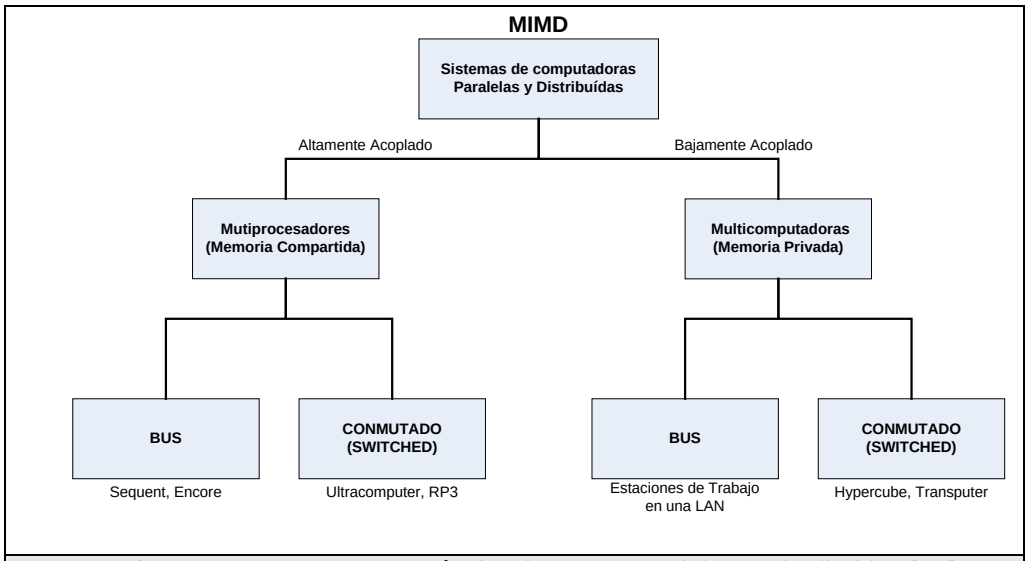


Figura 2.2.4.5-2. Taxonomía de Sistemas Paralelos y Distribuidos [F4]

A su vez, cada una de estas categorías puede ser dividida en base a la arquitectura de la red de interconexión. En la Fig. 2.2.4.5-2 se describen estas dos categorías como de **bus** y **conmutada**. A través de bus se refiere a que existe una red, una placa, un bus, un cable u

otro medio que conecta todas las máquinas. Conmutada en cambio, significa que no existe un medio que ayude a conectar todas las máquinas, en lugar de eso existen cables individuales interconectados de máquina a máquina.

Otra manera de dividir los sistemas MIMD, es según el acoplamiento de las máquinas que lo componen. Como vemos en la Fig. 2.2.4.5-2, existen sistemas donde las máquinas están **altamente acopladas** y otros donde están **bajamente acopladas**.

Un sistema altamente acoplado es aquel en que los componentes tienden a estar conectados de manera muy cercana entre sí. Estos sistemas suelen ser muy confiables; poseen un alto ancho de banda lo que hace que el retardo experimentado en la transferencia de mensajes entre CPUs sea muy corto, y la tasa de transferencia de datos sea muy alta. Por otro lado, un sistema bajamente acoplado es aquel donde los componentes tienden a estar distribuidos. El retardo tiende a ser alto y la tasa de transferencia de datos baja. La confiabilidad de estos sistemas yace en que si un componente falla, éste no afectará la funcionalidad de los otros componentes.

2.2.4.5.1 *Multiprocesadores basados en bus*

En este tipo de sistemas, todos los CPUs son conectados a un solo bus [REF.25], véase la Fig. 2.2.4.5.1-1. La memoria y periféricos, también son conectados al bus.

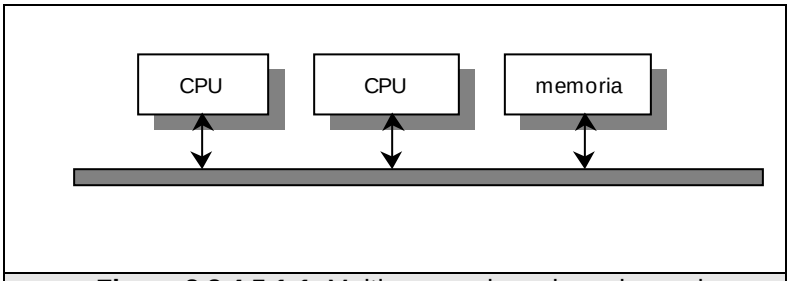


Figura 2.2.4.5.1-1. Multiprocesadores basados en bus

Una configuración sencilla pudiera ser el tener una placa madre de alta velocidad donde los CPUs y tarjetas de memoria puedan ser colocadas. Un bus normal tiene 32 ó 64 líneas de dirección, 32 ó 64 líneas de datos, y tal vez 32 ó más líneas de control; todas éstas operando en paralelo.

Si el CPU A escribe una palabra en la memoria, el CPU B puede leer dicha palabra inmediatamente después; una memoria que cumple con esta propiedad se la denomina memoria coherente.

El problema con este tipo de esquema es que el bus tiende a sobrecargarse rápidamente, haciendo que el rendimiento decaiga de

manera considerable. La solución sería colocar una memoria caché entre el CPU y el bus, tal como se muestra en la Fig. 2.2.4.5.1-2.

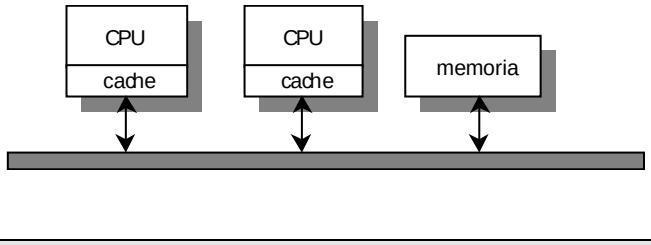


Figura 2.2.4.5.1-2. Multiprocesadores basados en bus con memoria caché

La función de la memoria caché es la de almacenar las regiones de memoria accedidas recientemente. De esta manera, el CPU debe acceder a la memoria a través del bus sólo cuando las regiones que solicite no estén en su caché. Si el tamaño de la caché es lo suficientemente grande la probabilidad de éxito[1] o llamada tasa de logro será alta; lo que ocasiona que el tráfico en el bus por CPU baje considerablemente; esto permite que el sistema pueda albergar una cantidad mayor de CPUs. Tamaños de memoria caché de 64K a 1M son muy comunes, las cuales a menudo brindan una tasa de logro mayor al 90%.

[1] Probabilidad de que la región solicitada se encuentre en la caché.

El problema que surge ahora es qué pasa si dos CPUs acceden a la misma región de memoria al mismo tiempo, cargan dichos datos en sus respectivas cachés para hacer futuras referencias a la misma. Suponga que el CPU A realiza una escritura que modifica un dato de la memoria; dicha modificación es local, esto es, sólo se da lugar en su respectiva caché. De esta forma cuando un CPU B lee la memoria, éste no cargará los nuevos datos que han sido modificados por A; lo cual conlleva a un problema de incoherencia de memoria. Una solución es usar una caché de tipo write-through[1]. En este caso, cualquier escritura es hecha no solo en la caché, sino que también es enviada a la memoria principal a través del bus. Aunque estas escrituras a memoria generan tráfico en el bus, las lecturas no lo harán; ya que sólo acceden al bus cuando los datos que se necesitan no están en la caché.

Esta solución planteada no es suficiente, ya que alguna caché podría tener almacenada una copia de algún dato que haya sido modificado. Este problema lo podemos resolver si cada caché monitorea constantemente el bus. Si una caché nota que se ha hecho una escritura en una dirección presente en su caché, ésta puede realizar

[1] También llamada caché de escritura continua.

dos acciones: remover el valor de su caché o actualizarlo con el nuevo valor presente en el bus[1]. Este tipo de caché es conocida como snoopy cache[2], ya que siempre se encuentra alerta de lo que acontece en el bus. Casi todos los sistemas de multiprocesadores basados en bus usan esta técnica o al menos una cercana a ella. Gracias a ella, es posible colocar alrededor de 32 o quizás aún 64 CPUs en un solo bus.

2.2.4.5.2 *Multiprocesadores conmutados*

Cuando se requiere construir un sistema multiprocesador de más de 64 procesadores, se debe usar una forma diferente de conectar los CPUs con las memorias. Una posibilidad es el dividir la memoria en módulos y conectarlas con los CPUs a través de un **crossbar switch** o malla conmutada, tal como se muestra en la Fig. 2.2.4.5.2-1. Cada CPU y cada memoria tienen una vía de comunicación por medio de un **crosspoint switch** o punto de conmutación. La ventaja de esta configuración es que permite que muchos CPUs puedan acceder a la memoria al mismo tiempo, sin embargo cuando esto sucede, uno de ellos deberá esperar a que el otro termine de usarla [REF.25].

---

[1] Cualquiera de las dos acciones que se ejecuten logrará una coherencia en la memoria.

[2] También es conocida como caché curiosa.

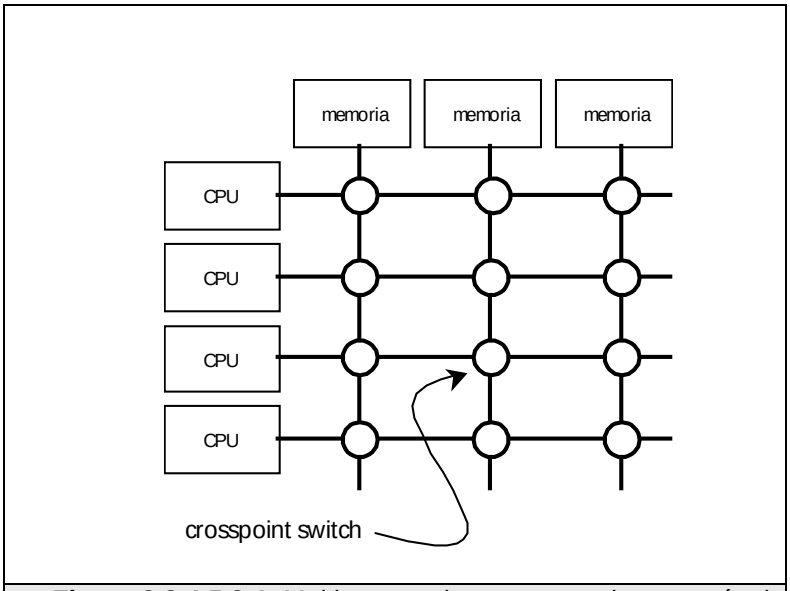


Figura 2.2.4.5.2-1. Multiprocesadores conectados a través de un crossbar switch.

La desventaja de usar una malla conmutada, es que puede llegar a ser muy costosa: conectar *n* CPUs con *n* módulos de memoria, requiere usar n^2 conmutadores o puntos de conmutación. Para reducir el número de conmutadores y mantener la misma conectividad, se requiere incrementar el número de estados en los que pueda estar cada conmutador. Lo que nos da como resultado la **red omega**, véase la Fig. 2.2.4.5.2-2, en la cual para un sistema de *n* CPUs y *n* módulos de memoria, requiere *n/2* conmutadores, cada uno de ellos con $log_2\ n$ estados. Aunque resulte mejor que tener n^2 conmutadores aún podemos decir que es una cantidad alta. Más aún, como cada conmutador posee más estados, se incrementa

notablemente el retardo. Con 1024 CPUs y memorias, se tendría que pasar a través de 10 estados de conmutación para acceder a la memoria, y diez más para regresar. Para poder evitar este retardo, se puede usar una memoria de acceso jerárquico: cada CPU puede acceder a su propia memoria de manera más rápida, mientras que el acceso a la memoria de otro CPU resulte más lento. Esta arquitectura es conocida como Memoria de Acceso No Uniforme o NUMA[1]. Esta arquitectura a pesar de proveer el mejor tiempo de acceso, se requiere de programación más compleja con el fin de maximizar los accesos a la memoria local.

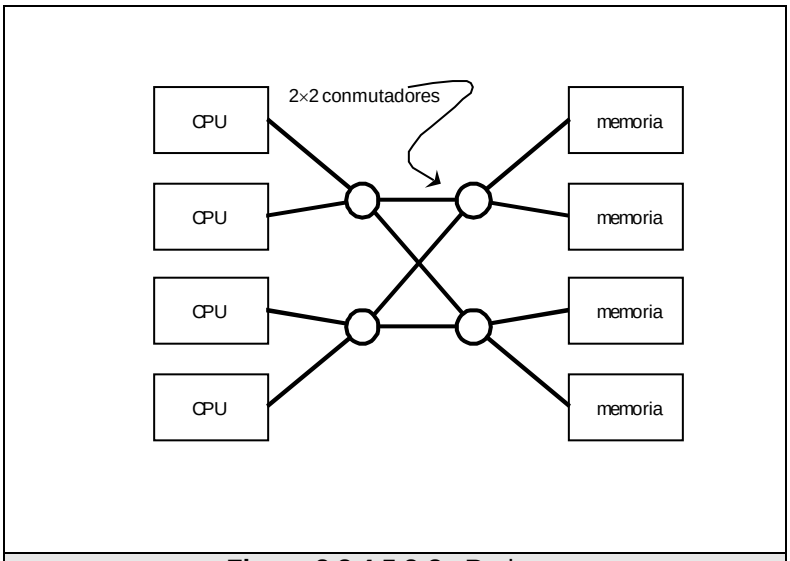


Figura 2.2.4.5.2-2. Red omega.

[1] Por sus siglas en inglés: Non-Uniform Memory Access.

En cuanto a los Multiprocesadores en general, se puede decir que para construir un gran sistema altamente acoplado con memoria compartida es posible, pero resulta demasiado costoso y difícil hacerlo.

2.2.4.5.3 *Multicomputadoras basadas en bus*

Una multicomputadora basada en bus es relativamente fácil de diseñar, el hecho está en que no se necesita lidiar con una memoria compartida: cada CPU simplemente tiene su propia memoria local [REF.25]. Sin embargo, sin la ayuda de una memoria compartida, algún otro mecanismo de comunicación es necesario para comunicar y sincronizar los procesos.

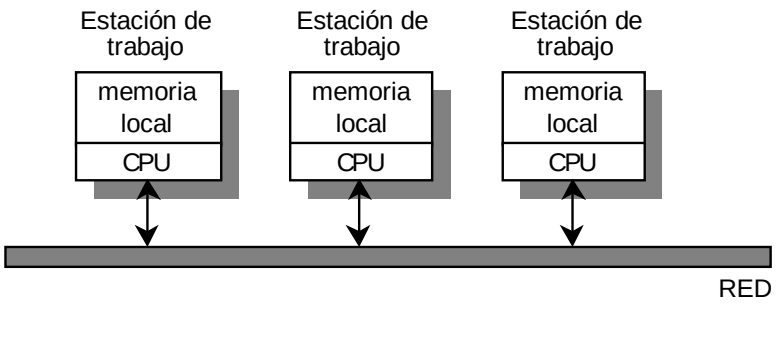


Figura 2.2.4.5.3-1. Multicomputador basado en bus, que consiste en estaciones de trabajo conectadas sobre una LAN.

Como vemos en la Fig. 2.2.4.5.3-1, la comunicación entre los CPUs podría ser a través de una LAN. El tráfico a través de una LAN es mucho menor que el tráfico provocado por los accesos a memoria de los sistemas de multiprocesadores basados en bus; de esta forma más CPUs podrían agregarse al sistema[1]. Este esquema de tener estaciones de trabajo conectadas a través de una red de área local, es muy común en los sistemas de multicomputadoras basadas en bus.

2.2.4.5.4 *Multicomputadoras conmutadas*

En este esquema de multicomputadoras conmutadas, cada CPU tiene acceso directo y exclusivo a su propia memoria privada [REF.25]. En la Fig. 2.2.4.5.4-1 se muestran las dos topologías más comunes, grid[2] e hipercubo.

---

[1] Un bus de sistema por lo regular tiene velocidades de entre 300Mbps a 1Gbps. En cambio, las velocidades más comunes en una LAN se encuentran en el intervalo de 10Mbps a 1Gbps.

[2] Algunas veces también es conocido como Grilla.

Un grid es muy fácil de entender e implementar, y están optimizados para problemas de naturaleza bidimensional, como el análisis de imágenes o visión por computadora.

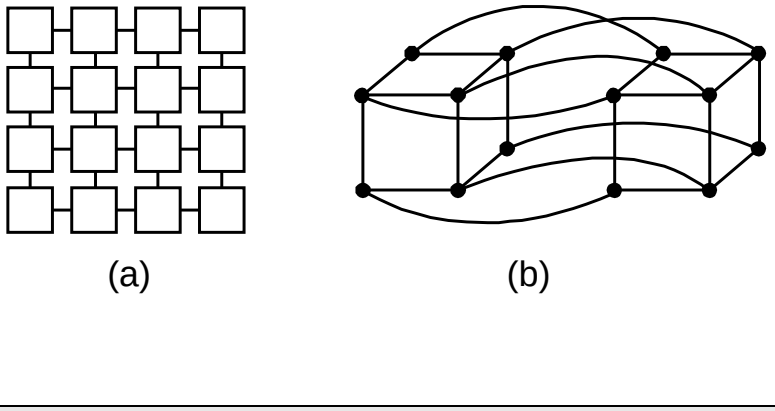


Figura 2.2.4.5.4-1. (a) Grid, (b) Hipercubo

Un hipercubo, es un cubo *n-dimensional*. El hipercubo de la Fig. 2.2.4.5.4-1 (b) es un cubo de 4 dimensiones; éste puede ser visto como dos cubos conectados entre sí, cada uno de ellos con 8 vértices y 12 aristas. Cada vértice representa un CPU y cada arista una conexión entre estos. Finalmente, los vértices correspondientes de cada cubo son interconectados entre sí.

2.2.5 Arquitecturas de procesamiento paralelo

La idea básica detrás del procesamiento paralelo es que varios dispositivos o procesadores, ejecutando de manera simultánea y

coordinadamente las tareas, pueden rendir más que un único dispositivo.

Si bien el procesamiento paralelo ofrece una ventaja definitiva en cuanto a costos, su principal beneficio, la escalabilidad, puede ser difícil de alcanzar. Esto se debe a que conforme se añaden procesadores, las disputas por los recursos compartidos se intensifican.

Algunas alternativas de arquitecturas de procesamiento paralelo enfrentan este problema fundamental, con diferentes resultados, entre las que se puede mencionar:

- Multiprocesamiento simétrico.
- Procesamiento masivamente paralelo.
- Procesamiento paralelo escalable.

2.2.5.1 Multiprocesamiento Simétrico (SMP)

El Procesamiento Simétrico[1], tiene un diseño simple, efectivo y económico. En SMP, muchos procesadores comparten la misma memoria RAM y bus del sistema [REF.26].

[1] Symmetric Multiprocessing.

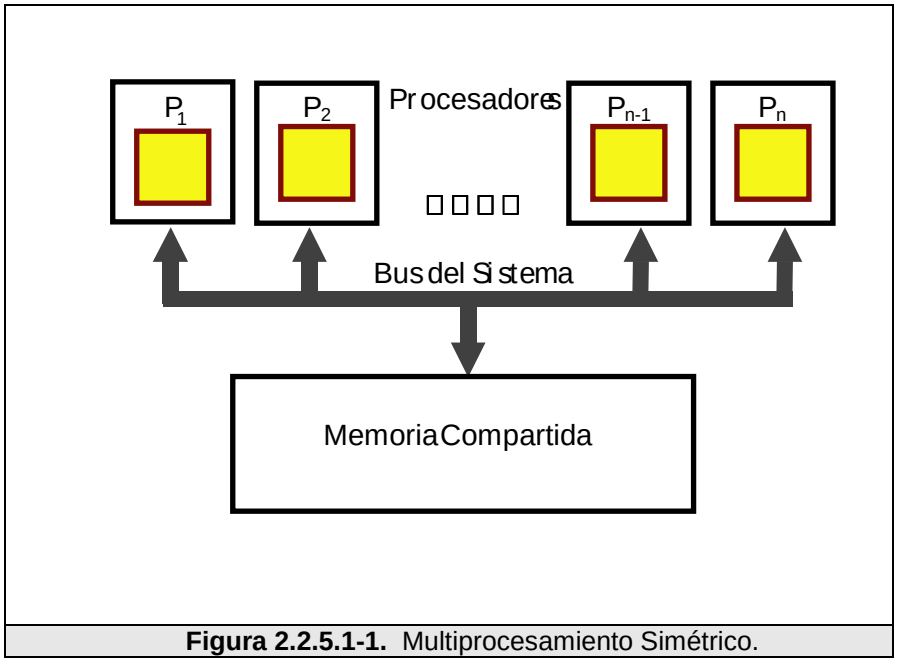


Figura 2.2.5.1-1. Multiprocesamiento Simétrico.

La presencia de un solo espacio de memoria simplifica tanto el diseño del sistema físico o *hardware*, como la programación de las aplicaciones o *software*. Esa memoria compartida permite que un Sistema Operativo con Multiconexión distribuya las tareas entre varios procesadores, o que una aplicación obtenga toda la memoria que necesita para una simulación compleja. La memoria globalmente compartida también vuelve fácil la sincronización de los datos.

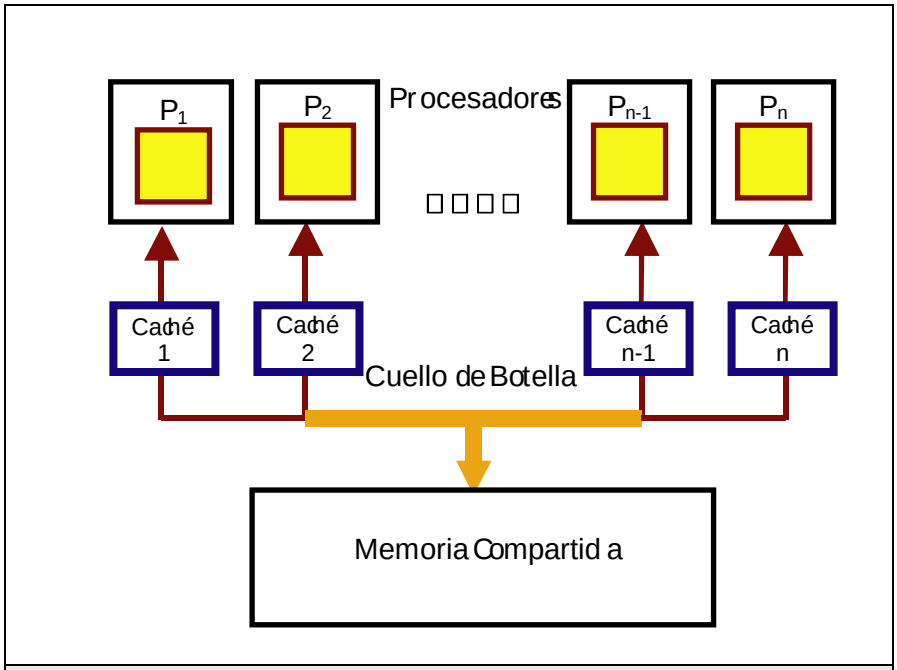


Figura 2.2.5-1-2. Multiprocesamiento Simétrico usando memoria caché.

SMP es uno de los diseños de procesamiento paralelo más maduros. Sin embargo, la memoria global contribuye al problema más grande de SMP: conforme se añaden procesadores, el tráfico en el bus de memoria se satura. Al añadir memoria caché a cada procesador se puede reducir algo del tráfico en el bus.

Al manejarse ocho o más procesadores, el cuello de botella se vuelve crítico, inclusive para los mejores diseños, por lo que SMP es considerada una tecnología poco escalable.

2.2.5.2 Procesamiento Masivamente Paralelo (MPP)

También llamado Arquitectura de Procesadores Masivamente Paralelos[1]. Es una arquitectura computacional de alto rendimiento. Para evitar los cuellos de botella en el bus de memoria, MPP no utiliza memoria compartida; en su lugar, distribuye equitativamente la memoria RAM entre los procesadores semejante a una red[2].

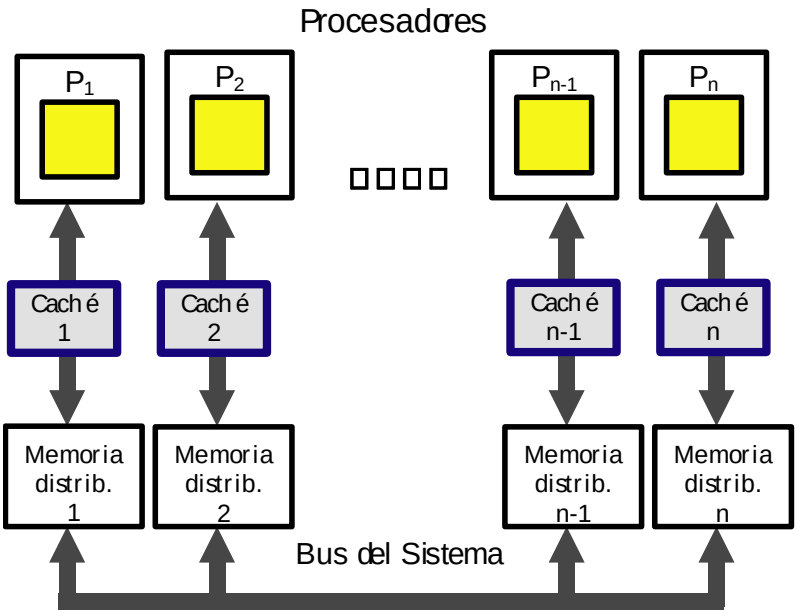


Figura 2.2.5.2-1. Procesamiento Masivamente Paralelo - Procesadores Masivamente Paralelos.

[1] Masively Paralel Processing.

[2] Cada procesador con su memoria distribuida asociada es similar a un computador dentro de una red de procesamiento distribuido.

Para tener acceso a las áreas de memoria fuera de su propia RAM (memoria libre no empleada por los otros procesadores), los procesadores utilizan un esquema de **paso de mensajes** análogo a los paquetes de datos en redes. Este sistema reduce el tráfico del bus, debido a que cada sección de memoria interactúa únicamente con aquellos accesos que le están destinados, en lugar de interactuar con todos los accesos a memoria, como ocurre en un sistema con memoria distribuida. Esto permite la construcción de sistemas MPP de gran tamaño, con cientos y aún miles de procesadores, por lo que MPP es una tecnología altamente escalable [REF.26].

La desventaja de los MPPs, desde el punto de vista tecnológico, es que la programación se vuelve difícil, debido a que la memoria se rompe en pequeños espacios separados. Sin la existencia de un espacio de memoria globalmente compartido, ejecutar una aplicación que requiere una gran cantidad de RAM (comparada con la memoria local), puede ser difícil. La sincronización de datos entre tareas ampliamente distribuidas también se complica, particularmente si un mensaje debe pasar por muchos componentes de hardware hasta alcanzar la memoria del procesador destino.

Escribir una aplicación MPP también requiere estar al tanto de la organización de la memoria manejada por el programa. Donde sea necesario, se deben insertar comandos de paso de mensajes dentro del código del programa. Además de complicar el diseño del software, tales comandos pueden crear dependencias de hardware en las aplicaciones, esto es, sólo podría funcionar en equipos de cierta marca y modelo. Sin embargo, la mayor parte de vendedores de computadores han salvaguardado la portabilidad de las aplicaciones adoptando ya sea un mecanismo de dominio público para paso de mensajes como PVM, o el estándar MPI.

El costo de las soluciones basadas en MPP es mucho más alto que el costo por procesador de las soluciones con memoria compartida, por lo que su uso sólo se justifica cuando la necesidad de procesamiento es muy alta.

2.2.5.3 Procesamiento Paralelo Escalable (SPP)

La tercera arquitectura paralela, el Procesamiento Paralelo Escalable[1], es un híbrido de SMP y MPP, que utiliza una memoria jerárquica de dos niveles para alcanzar la escalabilidad. La primera

[1] Scalable Parallel Processing.

capa consiste de componentes de memoria distribuida que son esencialmente parte de sistemas MPP completos, con múltiples nodos[1], y el segundo nivel de memoria está globalmente compartido al estilo SMP.

Se construyen sistemas SPP grandes interconectando dos o más nodos a través de la segunda capa de memoria, de modo que esta capa aparece, lógicamente, como una extensión de la memoria individual de cada nodo.

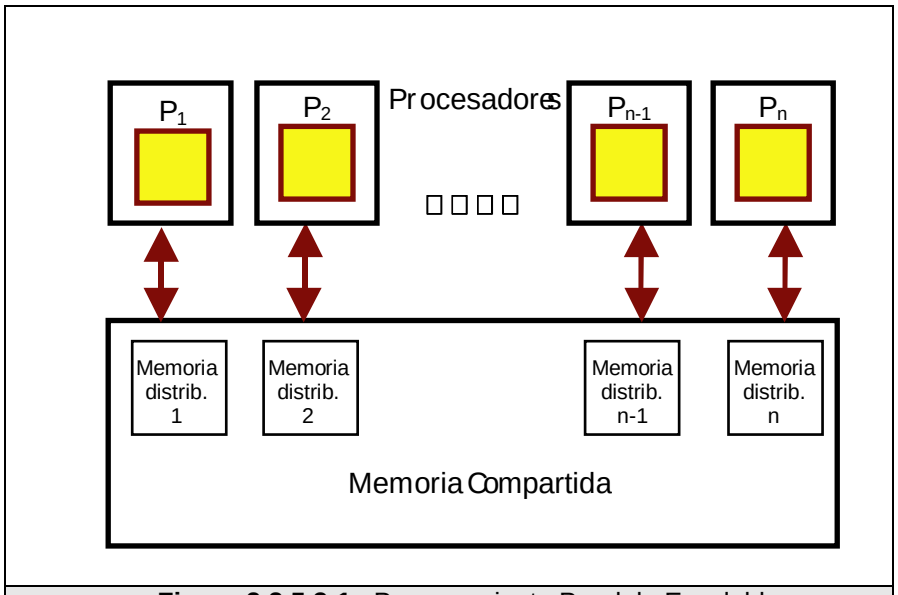


Figura 2.2.5.3-1. Procesamiento Paralelo Escalable.

[1] Un nodo está compuesto por un procesador más una memoria distribuida

La memoria de dos niveles reduce el tráfico de bus debido a que solamente ocurren actualizaciones para mantener coherencia de memoria. Por tanto, SPP ofrece la facilidad de programación del modelo SMP, a la vez que provee una escalabilidad similar a la de un diseño MPP.

2.2.6 Modelos de programación de sistemas paralelos

La programación paralela difiere mucho de la secuencial. La idea básica del procesamiento en paralelo es que alguna clase de tarea se divida en procesos que cooperen entre sí para resolver un problema común [REF.27].

La programación paralela es muy importante, ya que ésta nos ayuda a tomar decisiones de diseño sobre la arquitectura a utilizar, además de evaluar el desempeño de un sistema paralelo. Existen algunos modelos para la programación paralela, entre los más comunes tenemos:

- Modelo de memoria compartida.
- Modelo basado en hilos.
- Modelo de paso de mensajes.
- Modelo de paralelización de datos.
- Modelos híbridos.

Los modelos de programación paralela existen como una abstracción del la arquitectura de hardware y memoria. Aunque no resulte muy evidente, estos modelos no son específicos para una máquina o arquitectura de memoria en particular. Más aún, teóricamente cualquier modelo podría ser implementado en cualquier arquitectura.

La decisión de qué modelo usar se basa en la combinación de lo que esté disponible y una elección personal. En general, no existe “el mejor modelo”, aunque ciertamente algunas implementaciones resultan mejores que otras.

Existen dos estilos fundamentales para la construcción de programas paralelos:

- **Usando flujo de control**[1].Se crean procesos diferentes que operan sobre sus propios datos. La funcionalidad del programa se divide en procesos.
- **Usando paralelismo de datos**[2]. Se crean procesos iguales que aplican las mismas instrucciones sobre datos propios. Los datos del programa se dividen en procesos similares.

[1] También conocido como Control Flow.

[2] También conocido como Data Parallel.

2.2.6.1 Modelo de memoria compartida

En el modelo de programación con memoria compartida, los procesos comparten espacios de dirección comunes, los cuales son leídos y escritos de manera asincrónica [REF.28].

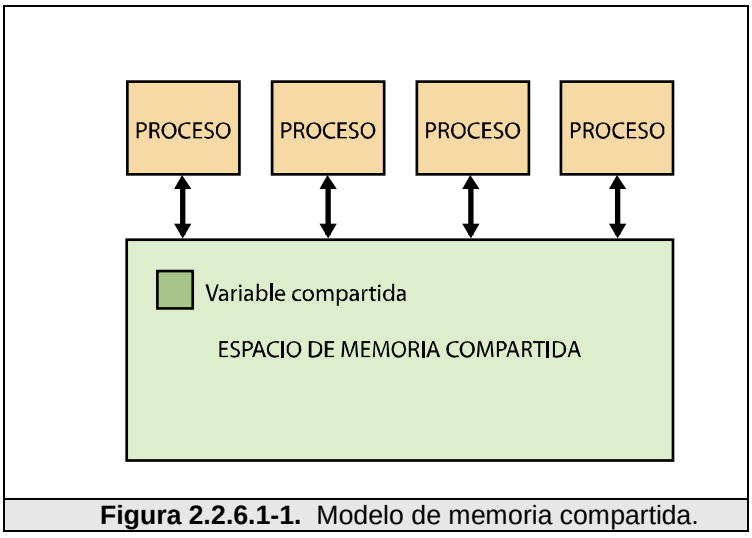


Figura 2.2.6.1-1. Modelo de memoria compartida.

Los procesos intercambian mensajes sobre la memoria compartida o actúan coordinadamente sobre los datos residentes en ella.

Lógicamente los procesos no pueden operar simultáneamente sobre la memoria compartida; por esto, varios mecanismos como *bloqueo*[1] o *semáforo*[1] son utilizados para controlar el acceso a dicha memoria.

[1] El proceso bloquea temporalmente el acceso a la memoria compartida mientras la esté usando.

Una ventaja de este modelo, desde el punto de vista del programador es que la idea de “propietario del dato” no existe, de esta forma no hay la necesidad de especificar de manera explícita la comunicación de datos entres tareas, haciendo que la programación sea simplificada. Una desventaja en términos de rendimiento, es que se vuelve difícil de entender y administrar datos localmente.

En plataformas de programación usando memoria compartida, el compilador traduce las variables del programa en direcciones de memoria globales. Ejemplo de este tipo de plataformas es KSR ALLCACHE.

La arquitectura **KSR ALLCACHE** presenta al usuario un modelo de programación con memoria compartida, aunque ésta sea implementada sobre una memoria física distribuida. Hay dos características fundamentales de esta arquitectura que son usadas en la codificación. Primero, todos los datos son almacenados en un único espacio de dirección global. Esto es, todos aquellos datos que no son especificados como privados son automáticamente compartidos por todos los procesadores, simplemente

[1] Existen variables de control que habilita o no el acceso de un proceso a la memoria compartida.

accediéndolos como una variable global o un puntero. Segundo, toda "subpágina de memoria" de 128 bytes[1] puede ser bloqueada automáticamente por el procesador, de esta forma éste sólo restringe el acceso a porciones de la memoria compartida o subpáginas. [REF.29]

2.2.6.2 Modelo basado en hilos

En el modelo de programación basado en hilos[2], un proceso puede tener múltiples trayectorias de ejecución concurrente. La resolución de un problema se puede realizar en más de una tarea simultáneamente ejecutando más de un proceso, a su vez un proceso puede hacer lo mismo ejecutando más de un hilo al mismo tiempo. Cada hilo es un flujo de control diferente que puede ejecutar sus instrucciones de forma independiente [REF.30].

Un hilo se compone de registros, una pila y algunos otros datos; el resto de la estructura de los procesos la comparten todos los hilos, es decir el espacio de direcciones, los descriptores de archivos, entre otros.

---

[1] La unidad básica de memoria compartida en una máquina.

[2] También conocidos como Threads en inglés.

Las aplicaciones que ejecutan varios hilos a la vez se las denomina Aplicaciones de Multihilos[1]. El desarrollo basado en multihilos es una forma de explotar el paralelismo en computadoras multiprocesador, ya que los diferentes hilos pueden ejecutarse en distintos procesadores simultáneamente sin ningún tipo de información especial en el código fuente, y menos aún del usuario final.

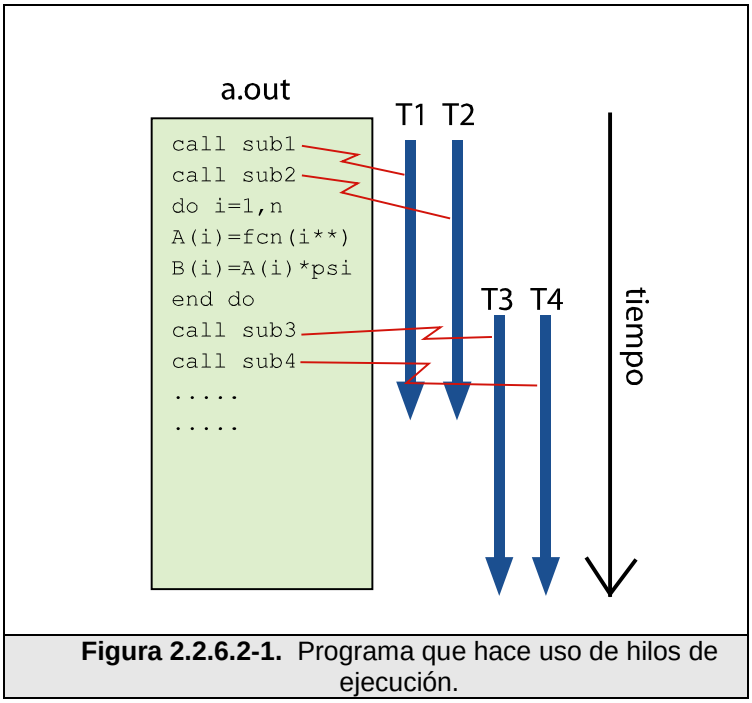


Figura 2.2.6.2-1. Programa que hace uso de hilos de ejecución.

La programación con hilos, también conocida como multiproceso, tiene la ventaja de poder trabajar de manera asíncrona. Esto permite que aquellos procesos que pueden requerir un tiempo más o menos

[1] Multithreads en inglés.

largo en llevarse a cabo, se pongan a trabajar “paralelamente” al proceso principal, de manera que la aplicación pueda retomar el control y así el usuario seguir trabajando con ella.

Quizás la analogía más simple que se puede utilizar para describir un hilo es el concepto de un programa que incluye un número determinado de subrutinas, véase la Fig. 2.2.6.2-1:

- El programa principal **a.out** está programado para ejecutarse en algún sistema operativo; éste para poder ejecutarse, carga todo lo necesario, tanto del sistema operativo como de los recursos del usuario.
- **a.out** al inicio ejecuta algo de trabajo secuencial; y luego de esto, crea dos hilos que son ejecutados de manera concurrente por el sistema operativo.
- Aunque cada hilo posee datos locales, también comparte todos los recursos que **a.out** posee. De esta manera se evita la sobrecarga asociada con la replicación de recursos de la aplicación para cada hilo. Además cada hilo toma ventaja de una vista de memoria global ya que comparte el espacio de memoria de **a.out**.

- El trabajo de un hilo es mejor descrito como una subrutina dentro de un programa principal. Cualquier hilo puede ejecutar alguna subrutina al mismo tiempo que otros hilos.
- La comunicación entre hilos se realiza a través de una memoria global. Esta requiere de esquemas de sincronización para asegurar que más de un hilo no este modificando la misma dirección global al mismo tiempo.

Desde la perspectiva del programador, las implementaciones de hilos por lo regular consisten en:

- Librerías de subrutinas llamadas desde un código fuente paralelo.
- Un conjunto de directivas de compilador incluidas en el código fuente serial o en código fuente paralelo.

En ambos casos, el programador es responsable de determinar todo el paralelismo. Los hilos se están convirtiendo en el mecanismo de programación paralela de los multiprocesadores de memoria compartida, ya que permiten explotar el paralelismo de dichas arquitecturas de una forma eficiente.

Las implementaciones de hilos no son de lo más recientes. Históricamente, los proveedores de hardware han implementado sus propias versiones de hilos. Estas implementaciones difieren mucho unas de otras, haciendo difícil para los programadores el desarrollo de aplicaciones basadas en hilos que sean portables.

Esfuerzos no combinados de estandarización han dado como resultado dos implementaciones de hilos completamente diferentes: POSIX[1] Threads y OpenMP.

POSIX Threads, es una librería de C implementada como *pthread.h* que cumple los estándares POSIX. Para su utilización se requiere de una codificación paralela, permitiéndonos trabajar con distintos hilos de ejecución o threads al mismo tiempo; los hilos creados con estas librerías son denominados **pthreads**. Muchos proveedores de hardware proveen pthreads en adición a su propia implementación de hilos [REF.31].

[1] POSIX es el acrónimo de Portable Operating System Interface, viniendo la X de UNIX con el significado de la herencia de la API (Se traduciría como Sistema Operativo Portable basado en UNIX).

Por otro lado, **OpenMP** es un API que soporta la programación paralela en lenguajes como C/C++ y Fortran, tanto en sistemas multiplataforma como en sistemas de memoria compartida. La mayoría de las construcciones en OpenMP son directivas de compilación o pragmas. La parte central de OpenMP es la paralelización de lazos a través de hilos [REF.32].

2.2.6.3 Modelo de paso de mensajes

Es probablemente el más extendido hoy en día en la programación de aplicaciones paralelas. El modelo de paso de mensaje demuestra las siguientes características:

- Un conjunto de tareas que usan su propia memoria local durante el cálculo; esto es, los procesos no comparten memoria. Múltiples tareas pueden residir en la misma máquina así como también a través de un número arbitrario de máquinas.
- Las tareas intercambian datos a través de comunicaciones de envío y recepción de mensajes, véase Fig. 2.2.6.3-1. Esto es, la comunicación se hace mediante operaciones explícitas de envío y recepción.
- La transferencia de datos usualmente requiere operaciones cooperativas a ser ejecutadas por cada proceso. Por ejemplo, la

operación de envío de mensaje deberá tener a la par una operación de recepción de mensaje.

El paso de mensaje es un modelo de *no-bloqueo*. Esto significa que un proceso puede continuar su ejecución inmediatamente después de que ha enviado un mensaje a otro proceso. El paso de mensajes puede ser asincrónico. El paso de mensajes asincrónico no implica que deba existir alguna clase de buffer, pero si requiere de una sincronización efectiva y robusta entre el emisor y el receptor. [REF.33]

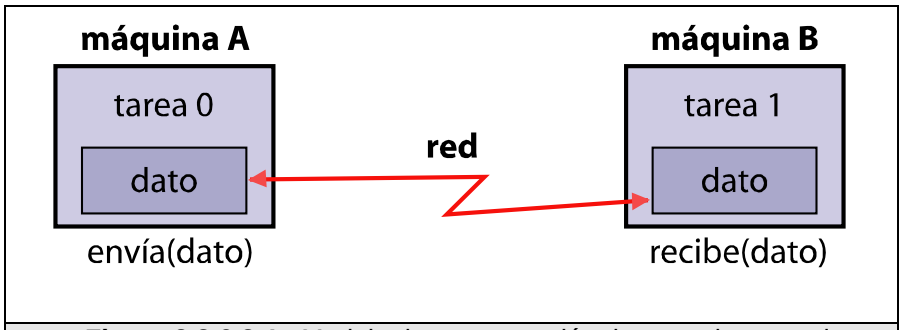


Figura 2.2.6.3-1. Modelo de programación de paso de mensajes.

El paso de mensaje podría ser o no confiable. Si es confiable, esto es un requerimiento que el sistema de comunicaciones subyacente deba cerciorarse que sea así. El paso de mensajes es apropiado para aplicaciones que son muy sensibles a latencias.

Desde una perspectiva de programación, las implementaciones de paso de mensaje por lo regular consisten en librerías de subrutinas que deben ser incluidas en el código fuente. El programador es responsable de determinar todo el paralelismo.

Con el pasar del tiempo una variedad de librerías de paso de mensaje han sido desarrolladas. Estas implementaciones difieren mucho unas de otras, haciendo difícil el desarrollo de aplicaciones portables.

En 1992, El forum MPI fue formado con el objetivo principal de establecer una interfase estándar para la implementación de paso de mensajes. En Noviembre de 1993 fue lanzada la **Interfase de Paso de Mensajes (MPI[1])**; un interfaz estándar para la realización de aplicaciones paralelas basadas en paso de mensajes, definida para C, C++ y FORTRAN.

Al diseñarse MPI, se tomaron en cuenta las características más atractivas de los sistemas existentes para el paso de mensajes, en vez de seleccionar uno solo de ellos y adoptarlo como el estándar.

---

[1] Message Passing Interface.

La meta de MPI, es el desarrollar un estándar que sea ampliamente usado para escribir programas que implementen el paso de mensajes. Por lo cual el interfaz intenta establecer para esto un estándar práctico, portable, eficiente y flexible.

En un ambiente de comunicación con memoria distribuida en la cual las rutinas de paso de mensajes son de nivel bajo; los beneficios de la estandarización son muy notorios. La principal ventaja al establecer un estándar para el paso de mensajes es la portabilidad y el ser fácil de utilizar [REF.34].

MPI es un sistema complejo, el cual comprende 129 funciones, de las cuales la mayoría tiene muchos parámetros y variantes.

Para arquitecturas de memoria compartida, las implementaciones de MPI no usan la red para la comunicación entre tareas; en lugar de eso, usan la memoria compartida por razones de rendimiento.

2.2.6.4 Modelo de paralelización de datos

Los modelos de paralelización de datos demuestran las siguientes características:

- Mucho del trabajo en paralelo se enfoca en ejecutar operaciones sobre un conjunto de datos. Este conjunto de datos por lo regular es organizado en una estructura común, como un arreglo o un cubo.
- Un grupo de tareas trabajan colectivamente sobre la misma estructura de datos, sin embargo, cada tarea trabaja sobre una porción diferente de dicha estructura.
- Las tareas ejecutan la misma operación en su porción de trabajo, por ejemplo: “sumar cuatro a cada elemento del arreglo”.
- Los datos son distribuidos a través de los procesadores.

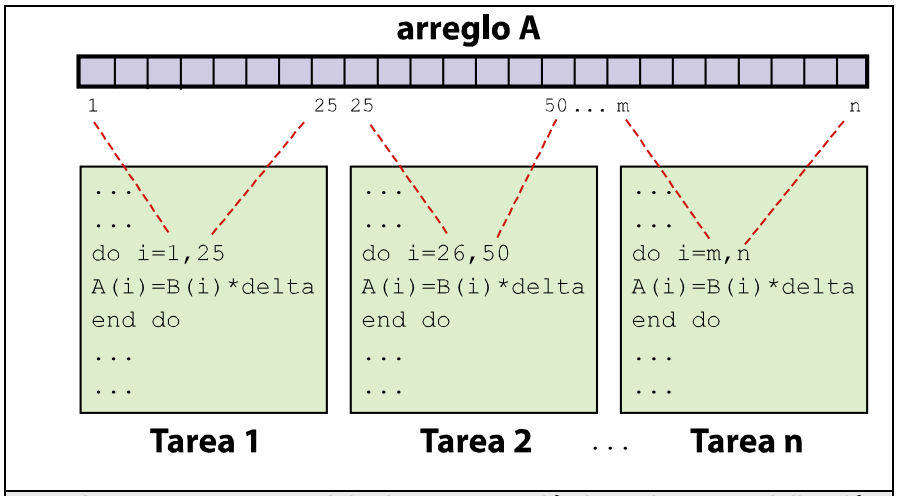


Figura 2.2.6.4-1. Modelo de programación basado en paralelización de datos.

En general, esta forma de cooperación se basa en un grupo de procesadores que efectúan operaciones sobre diferentes elementos de un conjunto de datos de forma simultánea, intercambiándose la

información globalmente antes de continuar. El intercambio de información puede realizarse a través de accesos a memoria compartida o mediante paso de mensajes, ya que esta cuestión no depende del modelo. La aplicación de este modelo está restringida a la resolución de problemas que impliquen el procesamiento en paralelo de datos con una estructura regular, como por ejemplo matrices [REF.35].

En arquitecturas de memoria distribuida, todas las tareas tienen acceso a la estructura a través de una memoria global. En arquitecturas que no tienen memoria distribuida, la estructura es dividida en “piezas”, las cuales residen en la memoria local de cada tarea.

La programación con este modelo es lograda codificando el programa con constructores de datos paralelos. Los constructores pueden llamar a librerías con subrutinas especiales para datos paralelos o, a directivas de compilación reconocidas por un compilador de datos paralelos.

Una plataforma de programación que nos ayuda a utilizar este modelo es **High Performance Fortran (HPF)**, el cual es una

extensión de Fortran 90[1] para proporcionar soporte a la programación paralela.

HPF contiene directivas para decirle al compilador como distribuir los datos agregados; así como constructores para datos paralelos. Funciona como directivas que pueden ser condicionalmente compiladas, algunos intrínsecos[2] y atributos que regulan el tipo de ejecución/alcance de datos de una subrutina. Pretende ignorar si la máquina sobre la que funciona es una multicomputadora o un multiprocesador [REF.37].

Las implementaciones de memoria distribuida para este modelo, generalmente hacen que el compilador convierta el programa en un código estándar con llamadas a librerías de paso de mensajes (usualmente MPI) para distribuir los datos a todos los procesos. Todos los pasos de mensaje entre los procesos se hacen de manera oculta al programador.

---

[1] Extensión estándar ANSI/ISO para Fortran 77. Fortran 77 es un lenguaje de propósito general, principalmente orientado a la computación matemática. Fortran es un acrónimo de FORmula TRANslator; y es considerado como uno de los primeros lenguajes de alto nivel.

[2] Tipos de datos u objetos propios del lenguaje.

2.2.6.5 Modelo Parallel Random Access Machine (PRAM)

El modelo PRAM [REF.37], es una extensión del modelo RAM, es uno de los modelos más conocidos para algoritmos paralelos.

P procesadores acceden a una memoria compartida global M. Cada procesador posee una memoria local propia y es identificado por un índice único; además, éste puede comunicarse con los demás procesadores a través de una memoria global compartida, véase Fig. 2.2.6.5-1. La ejecución sobre el modelo PRAM supone una secuencia de pasos con sincronización implícita de todos los procesadores. Cada procesador puede leer y procesar en su memoria local, o a su vez realizar intercambios con la memoria global [REF.38].

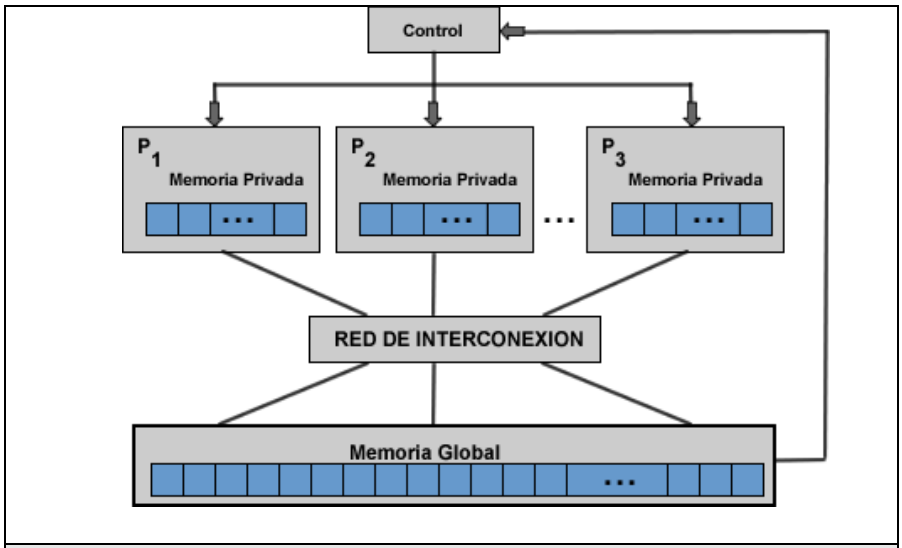


Figura 2.2.6.5-1. Modelo PRAM.

Cada procesador del conjunto puede leer un elemento de la memoria compartida hacia su memoria local o escribir un elemento de su memoria local hacia la memoria compartida; o en su defecto ejecutar cualquier operación RAM[1] sobre un dato contenido en su memoria local.

El modelo PRAM tal y como está definido es un modelo MIMD síncrono, lo que significa que los procesadores cuentan con distintos contadores de programa, los cuales pueden estar ejecutando diferentes instrucciones en un instante determinado, aunque todos ellos tardan el mismo número de ciclos en ejecutarlas. Sin embargo la mayoría de algoritmos PRAM explotan el modo SIMD.

El modelo PRAM opera de la siguiente manera:

1. Los datos de entrada están inicialmente en la memoria global del sistema.
2. Al inicio solamente un procesador está activo.
3. Durante cada paso de computación, cada procesador puede:
    a. Leer un valor de la memoria local o global
    b. Realizar una operación RAM

[1] Realizar alguna operación aritmético-lógica.

c. Escribir en la memoria global o local
d. Activar otro procesador

La forma de comunicación entre procesadores a través de una memoria global compartida da como resultado tres variaciones del modelo PRAM, todas ellas basadas en los permisos de acceso concurrente a dicha memoria global:

- EREW PRAM (Exclusive Read Exclusive Write), en donde sólo un procesador puede leer o escribir en una determinada posición de memoria a la vez.
- CREW PRAM (Concurrent Read Exclusive Write), en donde todos los procesadores pueden leer a la vez la misma posición global de memoria, pero solamente uno de ellos puede escribir en una posición de memoria a la vez.
- CRCW PRAM (Concurrent Read Concurrent Write), en donde múltiples procesadores pueden leer y escribir en una posición global de memoria de manera concurrente.

Se debe especificar una estrategia para solucionar los conflictos de acceso simultáneo en el modelo CRCW. Estas variantes adicionales son clasificadas como:

- COMMON CRCW PRAM, donde todos los valores escritos concurrentemente deben ser iguales.
- PRIORITY CRCW PRAM, donde el procesador que puede escribir durante una escritura concurrente es aquel procesador con mayor prioridad[1].
- ARBITRARY CRCW PRAM, donde el procesador que puede escribir durante una escritura concurrente es elegido al azar entre todos los procesadores involucrados en la escritura.
- COMBINING CRCW PRAM, donde el valor escrito es una combinación lineal de todos los valores que fueron escritos de manera concurrente, como por ejemplo escribir la suma de todos los valores. Esta combinación puede realizarse con cualquier operación asociativa y conmutativa que sea computable en un tiempo constante por una RAM.

El lenguaje más utilizado para programar sobre un modelo PRAM es **FORK**. Si bien FORK admite un modo sincrónico y otro asincrónico, dado el modelo PRAM se utilizan las primitivas sincrónicas. FORK está basado en ANSI C con extensiones para la administración de variables y espacios de dirección privados y compartidos, y para el

[1] Asumimos que el procesador con mayor prioridad es aquel que tiene el menor índice.

paralelismo anidado dinámico y estático por medio de constructores que dividen en grupos a los procesadores. El grupo establece el alcance de la memoria compartida y ejecución sincrónica. [REF.39] FORK es compatible con un grupo muy amplio de paradigmas para la programación en paralelo como paralelismo de datos, procesos asincrónicos coordinados con semáforos, pipelinig, paralelización de tareas e incluso paso de mensajes.

Las primitivas más conocidas son *fork* y *join*. En particular con *fork* se pueden crear procesos alternativos por las ramas de un IF y con *join* se genera una sincronización entre procesos mediante una barrera.

2.2.6.6 Modelo Bulk Synchronous Parallel (BSP)

El modelo BSP [REF.40] o Modelo Paralelo Síncrono Voluminoso fue propuesto por Valiant [REF.41] en 1990. Además de ser uno de los modelos realísticos más importantes, fue uno de los primeros a considerar los costes de comunicación y abstraer las características de una máquina paralela en un pequeño número de parámetros. El objetivo principal de este modelo es servir de modelo puente entre las necesidades de hardware y software en la computación paralela. Según Valiant, este es uno de los factores responsables del éxito del modelo secuencial de Von Neumann.

Una máquina BSP consiste de un conjunto de *p* procesadores con memoria local, comunicándose a través de algún medio de interconexión, controlados por un *ruteador* y con facilidad de sincronización global. Un algoritmo BSP consiste de una *secuencia de superpasos* separados por barreras de sincronización, como se muestra en la Fig. 2.2.6.6-1.

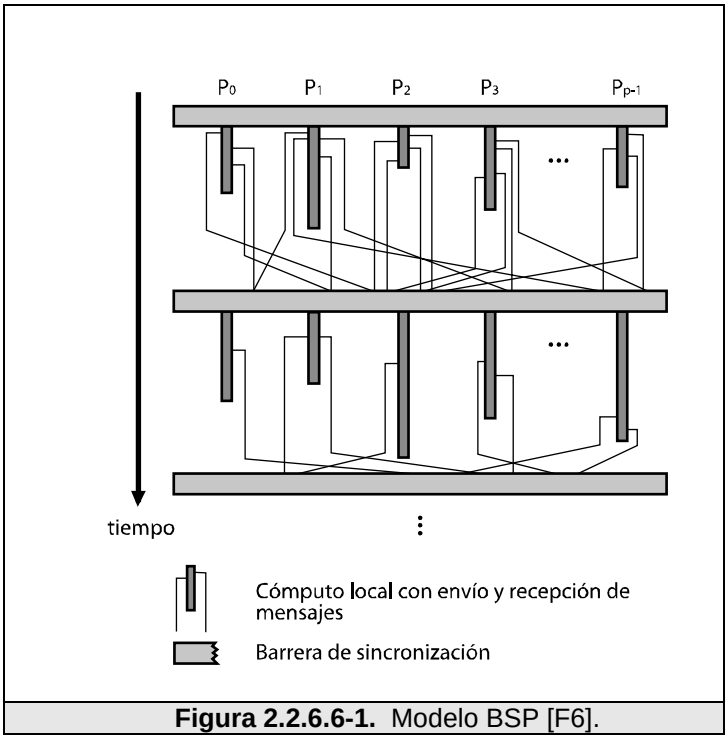


Figura 2.2.6.6-1. Modelo BSP [F6].

En un *superpaso*, a cada procesador se le atribuye un conjunto de operaciones independientes, consistiendo de una combinación de pasos de cómputo, usando datos disponibles localmente en el inicio

del superpaso, y pasos de comunicación, a través de instrucciones de envío y recepción de mensajes.

En este modelo, una *relación h* en un superpaso corresponde al envío y/o recepción de, a lo mucho, *h mensajes* en cada procesador. Un mensaje enviado en un superpaso será recibido solamente en el próximo superpaso.

Los parámetros del modelo BSP son los siguientes:

- n: tamaño del problema;
- p: número de procesadores disponibles, cada uno con su propia memoria local;
- L: el tiempo mínimo entre dos pasos de sincronización. También es llamado parámetro de periodicidad o de latencia de un superpaso;
- g: es la capacidad computacional dividida por la capacidad de comunicación de todo el sistema, o sea, la razón entre o número de operaciones de cómputo realizadas en una unidad de tiempo por el número de operaciones de envío e recepción de mensajes. Este parámetro describe la tasa de eficiencia de computación y comunicación del sistema.

Los parámetros *L* y *g* son usados para calcular los costos de comunicación en el tiempo de ejecución de un algoritmo BSP. El parámetro L representa el costo de sincronización, de tal forma que cada operación de sincronización contribuye con L unidades de tiempo al tiempo total de ejecución. L También puede ser visto como la latencia de comunicación, para que los datos recibidos puedan ser accedidos sólo por el siguiente superpaso.

La capacidad de comunicación de una red de computadores esta relacionada con el parámetro *g*. A través de este parámetro podemos estimar el tiempo tomado para el intercambio de datos entre procesadores. Si el número máximo de mensajes enviados por algún procesador durante un intercambio es *h*, entonces serían necesarias hasta *g.h* unidades de tiempo para realizar el intercambio [REF.42].

En la práctica, el valor de *g* es medido de forma experimental o estimado. De modo más formal el tiempo total de ejecución de un superpaso es igual a $w_i + g.h_i + L$, donde w_i es el monto máximo de trabajo local ejecutado por un procesador durante un *superpaso*, y h_i es el máximo número de mensajes enviados o recibidos por un procesador.

Asumamos que el algoritmo BSP posee un total de T superpasos, sea la suma de todos los valores de w y h de cada superpaso.

$$W = \sum_{i=0}^{T} w_i \quad \text{y} \quad H = \sum_{i=0}^{T} h_i$$

Entonces, el coste total de un algoritmo BSP es $W + g.h + L.T$. El valor de W representa el total de cómputos locales y el valor de H representa el volumen total de comunicación.

Entre las ventajas y desventajas de este modelo tenemos las siguientes:

1. Comparado con los modelos de red, BSP agrega simplificación y portabilidad.
2. Comparado con el modelo PRAM es más complejo y requiere medidas de los parámetros g, h y L, de cuya precisión dependerán las estimaciones.
3. Los superpasos simplifican el análisis, aseguran que no haya deadlock[1] pero agregan sobrecarga.

[1] Condición causada por un error de software en donde dos o más procesos en un sistema concurrente son interrumpidos indefinidamente.

4. Si bien hay costos de comunicación considerados, la localidad no se toma en cuenta, no es lo mismo comunicar un procesador adyacente que uno en otra red.

Dado que BSP ha tenido un desarrollo importante como modelo para predicción de rendimiento en clases de aplicaciones, hay un importante desarrollo de bibliotecas, como por ejemplo **BSP Lib**.

BSP Lib tiene unas 20 funciones que permiten: conocer la identificación o el número de procesos activos, una función de sincronización por barrera, comunicación half-duplex[1]. Put y Get sobre memoria local o externa, y funciones de comunicación tipo envió/recepción.

Hay otros desarrollos y extensiones por ejemplo Oxford Toolset o Paderborn University (PUB).

[1] Comunicación de dos vías pero en un solo sentido. No se pueden enviar y recibir datos simultáneamente.

2.2.7 Programación paralela

2.2.7.1 Entender y paralelizar el problema

Indudablemente, el primer paso en el desarrollo de software paralelo es entender el problema que se desea resolver de manera paralela.

Antes de perder el tiempo en intentar desarrollar una solución paralela a un problema, se debe determinar si el problema es o no paralelizable.

El problema de buscar los factores de un número grande es un ejemplo de problema paralelizable, ya que se puede dividir el trabajo de tal forma que cada procesador busque un factor dentro de un rango determinado.

Un ejemplo de un problema no paralelizable es tratar de encontrar la serie Fibonacci usando la fórmula $F(k + 2) = F(k + 1) + F(k)$, ya que el cálculo de un término de la serie es totalmente dependiente del resultado del cálculo del término anterior.

Se deben identificar las regiones críticas del programa donde se use gran parte de la potencia del CPU:

- Conocer dónde se está haciendo la mayor parte del trabajo real. Usualmente la mayor parte de programas científicos y técnicos llevan a cabo su trabajo en pequeñas secciones específicas del programa.
- El análisis de perfiles y rendimiento puede ser de gran ayuda.
- Enfocarse en paralelizar estas regiones críticas e ignorar aquellas secciones que cuenten con poco uso de CPU.

Se debe identificar los cuellos de botella que pudiera haber en el programa:

- ¿Dentro del programa existen regiones desproporcionadamente lentas, o regiones que causan que el trabajo paralelo se detenga o retrase? Por ejemplo, operaciones de E/S son las que usualmente hacen liento a un programa.
- Se puede reestructurar el programa o usar un algoritmo diferente para reducir o eliminar aquellas regiones lentas innecesarias.

Se debe identificar los *inhibidores de paralelismo*. Una clase común de inhibidores es la dependencia de datos, como fue demostrado anteriormente en el ejemplo de la secuencia Fibonacci.

Se debe investigar otros algoritmos si es posible. Esta puede ser una sencilla pero importante consideración que se debe tener al momento de diseñar una aplicación paralela.

2.2.7.2 Pasos para ejecutar un programa en paralelo

Para ejecutar un programa en paralelo se deben seguir los siguientes pasos:

- *Dividir:* Consiste en dividir el trabajo entre los múltiples procesadores.
- *Comunicar*: Consiste en enviar las instrucciones a cada procesador y recibir los resultados calculados por los mismos.
- *Sincronizar*: Consiste armonizar y combinar los resultados enviados por cada procesador.

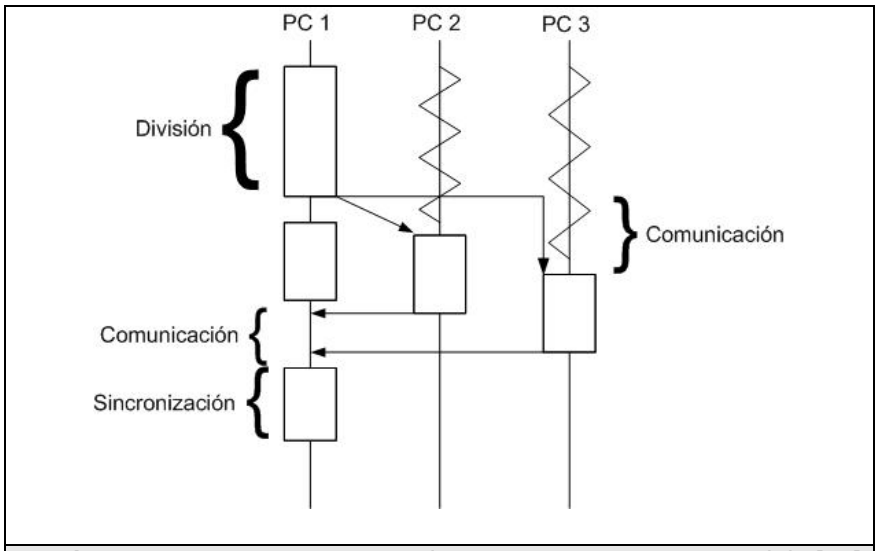


Figura 2.2.7.2-1. Pasos para ejecutar un programa en paralelo [F7].

2.2.7.2.1 Dividir

Uno de los primeros pasos en el diseño de programas paralelos es dividir el programa en “piezas” discretas de trabajo que puedan ser distribuidas sobre múltiples tareas. Esto se lo conoce también como descomposición o particionamiento.

Hay dos maneras básicas de hacer particionamiento de trabajo entre tareas paralelas: descomposición por dominio y descomposición funcional.

En la ***descomposición por dominio***, los datos asociados a un problema son descompuestos. De esta forma cada tarea paralela realiza trabajo sobre una porción de datos, como se muestra en la Fig. 2.2.7.2.1-1.

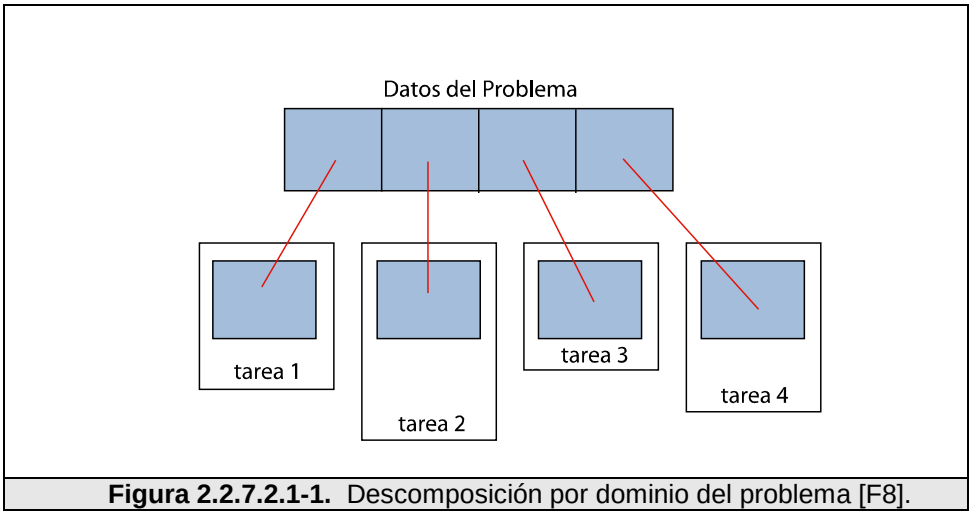


Figura 2.2.7.2.1-1. Descomposición por dominio del problema [F8].

Como vemos en la Fig. 2.2.7.2.1-2, existe una gran variedad de maneras de dividir los datos.

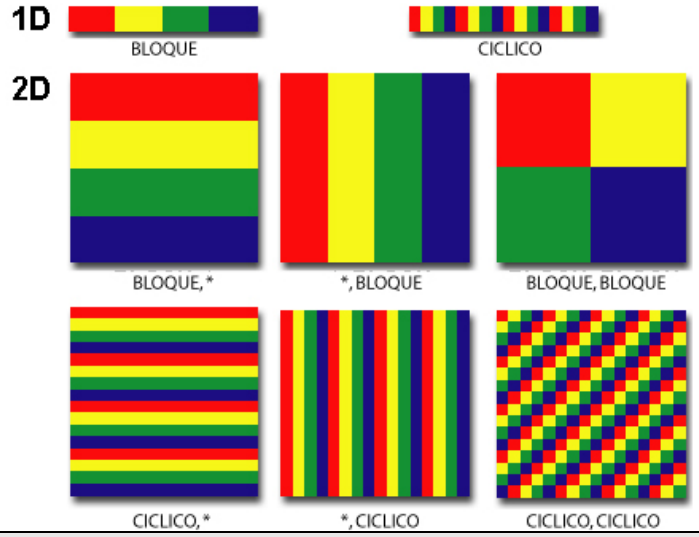


Figura 2.2.7.2.1-2. División de datos [F8].

La ***descomposición funcional***, se enfoca en el cómputo que debe ser realizado, antes que en los datos a ser computados.

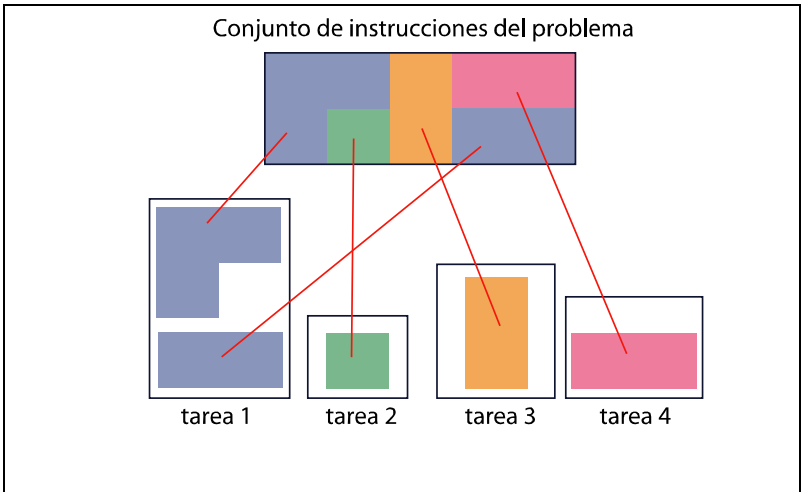


Figura 2.2.7.2.1-3. Descomposición funcional [F8].

El problema es dividido de acuerdo al trabajo que debe hacerse. Por lo tanto, cada tarea realiza una porción del trabajo total.

2.2.7.2.2 Comunicar

La necesidad de comunicación entre tareas, depende mucho de la naturaleza del problema:

- ***Tareas que no necesitan comunicación***: Ciertos tipos de problemas pueden ser descompuestos y ejecutados en paralelo, virtualmente sin la necesidad de que las tareas compartan datos entre sí. Por ejemplo, el procesamiento de una imagen donde cada píxel de dicha imagen deba ser invertido. Este trabajo puede ser distribuido muy fácilmente en múltiples tareas que actúan de manera independiente una de otra para realizar su porción de trabajo.

- ***Tareas que necesitan comunicación***: La mayoría de aplicaciones paralelas no son tan simples, y requieren que las tareas compartan datos entre sí. Por ejemplo, un problema de una representación 3-D de difusión de calor, requiere que una tarea conozca la temperatura calculada por su tarea vecina.

2.2.7.2.3 Sincronizar

Como ya se mencionó anteriormente, la sincronización consiste armonizar y combinar los resultados enviados por cada procesador. Existen tres tipos de sincronización:

- ***Sincronización de barrera***: Usualmente involucra todas las tareas involucradas. Cada tarea ejecuta su trabajo antes de alcanzar la barrera; una vez alcanzada, ésta se detiene o bloquea. Cuando la última tarea alcanza la barrera, todas las tareas están sincronizadas.

- ***Sincronización por candado/semáforo***: Puede involucrar un número indeterminado de tareas. Comúnmente usado para serializar o proteger el acceso a datos globales o una sección de código. Sólo una tarea puede usar el candado/semáforo al mismo tiempo. La primera tarea en tomar el candado, lo “setea”. Esta tarea ahora puede acceder de manera segura al dato o código que ha sido protegido. Las demás tareas pueden intentar adueñarse del candado pero deben esperar mientras la tarea anterior lo deje de usar.

- ***Sincronización por operaciones de comunicación***: Involucra sólo las tareas que ejecutan una operación de comunicación. Cuando una tarea ejecuta una operación de comunicación, alguna clase de coordinación con otra u otras tareas participantes en la comunicación es requerida. Por ejemplo, antes que una tarea ejecute una operación de envío de datos, deberá recibir una comunicación de parte de la otra tarea, de que está lista para recibir dichos datos.

2.3 Los sistemas distribuidos

2.3.1 Definición de los sistemas distribuidos

Existes muchas definiciones de sistemas distribuidos. Una de las primeras caracterizaciones de un sistema distribuido fue realizada en 1978 por Enslow [REF.43] que le atribuye las siguientes propiedades:

- Está compuesto por varios *recursos informáticos* de propósito general, tanto físicos como lógicos, los cuales pueden asignarse dinámicamente a tareas concretas.
- Estos recursos están *distribuidos físicamente*, y funcionan gracias a una red de comunicaciones.
- Hay un *sistema operativo* de alto nivel, que unifica e integra el control de los componentes.

- El hecho de la distribución es *transparente*, permitiendo que los servicios puedan ser solicitados especificando simplemente su nombre y no su localización.
- El funcionamiento de los recursos físicos y lógicos está caracterizado por una *autonomía coordinada*.

A pesar del tiempo transcurrido, esta definición sigue siendo, en esencia, válida. Así, para Coulouris [REF.44] un sistema distribuido es aquél que está compuesto por varios ordenadores autónomos conectados mediante una red de comunicaciones y equipados con programas que les permitan coordinar sus actividades y compartir recursos.

Bal también nos ofrece una definición muy similar de lo que es un sistema distribuido [REF.45]: "*Un sistema de computación distribuida está compuesto por varios procesadores autónomos que no comparten memoria principal, pero cooperan mediante el paso de mensajes sobre una red de comunicaciones*".

Y según Schroeder [REF.46], todo sistema distribuido tiene tres características básicas:

1. *Existencia de varios ordenadores*. En general, cada uno con su propio procesador, memoria local, subsistema de entrada/salida y quizás incluso memoria persistente.
2. *Interconexión*. Existen vías que permiten la comunicación entre los ordenadores, a través de las cuales pueden transmitir información.
3. *Estado compartido*. Los ordenadores cooperan para mantener algún tipo de estado compartido. De otra manera, puede describirse el funcionamiento correcto del sistema como el mantenimiento de una serie de invariantes globales que requiere la coordinación de varios ordenadores.

Sin embargo, para nuestros propósitos vamos a usar la definición propuesta por Tanembaum [REF.47]:

"Un sistema distribuido es una colección de computadoras autónomas e independientes, que aparecen frente a los usuarios del sistema como un solo computador".

Esta última definición tiene dos aspectos importantes. El primero de ellos tiene que ver con el hardware: las máquinas son autónomas. El

segundo tiene que ver con el software: los usuarios piensan en el sistema como un solo computador.

2.3.2 Características de los sistemas distribuidos

Además del hecho que los sistemas distribuidos proporcionan un alto desempeño a bajo costo, éste incluso puede llegar a alcanzar niveles de rendimiento fuera del alcance de una computadora normal.

A pesar de los progresos alcanzados en la tecnología de microprocesadores, existen limitaciones físicas en cuanto a las velocidades que pueden llegar a alcanzar los procesadores. Al permitir que un número de CPUs trabajen juntos para realizar la misma tarea, con el simple hecho de adicionar más CPUs se puede observar un notable incremento en el rendimiento del sistema; combinándolo con procesadores más rápidos, se pueden satisfacer necesidades que de otra manera no se lo podría hacer.

Adicionalmente, los sistemas distribuidos permiten un desarrollo incremental de poder de cómputo sin un reacondicionamiento completo del sistema.

Cuando existe la necesidad de actualizar, sin un sistema distribuido, una empresa necesita sustituir el sistema obsoleto por uno de mayor capacidad computacional. Esta operación además de ser difícil de realizar puede llegar a ser muy costosa. Un sistema distribuido puede potencialmente permitir que el usuario aumente el poder de cómputo, la comunicación, y las capacidades de almacenamiento del sistema con una transparencia completa hacia todos los clientes.

Todo esto conlleva a lo expresado por Tanembaum y que lo podemos denominar *virtualización de computadoras*. Un sistema distribuido permite que un usuario considere a la computadora como un dispositivo lógico, en lugar de un dispositivo físico. La actualización de un dispositivo físico no afecta la identidad lógica de la computadora, por lo tanto, es mucho más fácil y menos costosa. Otra ventaja que nos ofrece esta virtualización es que no solamente la adición de computadoras físicas es transparente a los usuarios, si no también su retiro. Con la adición de nodos el sistema experimenta lo que se denomina *escalabilidad de rendimiento*; y con el retiro de nodos el sistema experimenta una *degradación de rendimiento*.

Un sistema distribuido puede ser diseñado para ser tolerante a fallos. En un sistema distribuido tolerante a fallos, cuando una parte del

sistema falla, el sistema en general continúa funcionando. Ésta es una característica importante de un sistema distribuido, a veces aún más importante que todas las consideraciones del rendimiento y escalabilidad combinadas, especialmente en aplicaciones que ejecutan tareas críticas. En este tipo de sistemas, el mantenimiento se lo puede hacer sin necesidad de detenerlo por completo. El usuario puede hacer mantenimiento sobre una parte *fuera de línea*[1] del sistema, mientras que la otra parte continúa adquiriendo carga de trabajo. El empleo de un sistema tolerante a fallos puede aumentar considerablemente la confiabilidad y disponibilidad total de su infraestructura.

2.3.3 Heterogeneidad de los sistemas distribuidos

Los sistemas distribuidos ofrecen una gran ventaja al poder trabajar en un entorno de redes, al contrario de lo que ocurre con los sistemas MPP, por ejemplo, donde todos los procesadores tienen exactamente los mismos recursos, capacidades, software y velocidad de comunicación. En un sistema distribuido puede predominar la heterogeneidad (diversidad) de estos recursos tales como:

[1] Parte del sistema que se encuentra desconectada del sistema general.

- Arquitectura.
- Formato de datos.
- Velocidad de procesamiento.
- Carga de la máquina.
- Carga de la red.

Dado que el conjunto de computadoras que se puede encontrar en una red puede llegar a ser muy variado y diferir mucho en su arquitectura, podríamos encontrar computadoras de escritorio con diferentes tipos de procesadores, computadoras de alta velocidad, supercomputadoras, entre otras.

El formato de datos es un punto muy importante que debemos tomar en cuenta en la computación distribuida. Debido a la variedad de arquitecturas, las computadoras podrían incluso llegar a no entenderse si éstas no usan un formato de datos preestablecido en los mensajes que se envían entre sí por medio de la red.

Aunque el formato de datos de comunicación entre las computadoras sea el mismo, aún existe el problema de la diferencia de velocidades de procesamiento entre las máquinas que forman parte del sistema distribuido; debemos considerar que computadoras con mayor

capacidad computacional no esperan mucho tiempo por datos procesados por computadoras con menor velocidad de procesamiento.

Aunque el sistema distribuido se encuentre formado por máquinas idénticas, se debe tomar en cuenta la carga individual de la computadora, ya que el usuario podría estar ejecutando otras tareas en éste al mismo tiempo.

De igual forma que la carga de trabajo por computadora, el tiempo que toma en enviar un mensaje por la red puede variar dependiendo de la carga colocada por los demás usuarios de la red. Esta se debe tomar en cuenta con el fin de que una tarea no se quede esperando demasiado tiempo por un mensaje que debe llegar por medio de la red.

A pesar de todas estas consideraciones causadas por la heterogeneidad, las cuales se deben de tomar en cuenta en la computación distribuida, se tienen una gran cantidad de ventajas:

- Ahorro en costos de hardware, puesto que se usa los recursos de computadoras ya existentes.

- Se puede obtener un mejor rendimiento al asignar tareas a las computadoras con la arquitectura más apropiada.
- Se puede explotar al máximo el concepto de heterogeneidad, por ejemplo proveer el acceso a diferentes bases de datos y arquitecturas de procesador, para aquellas partes de la aplicación que pueden ser ejecutadas sólo en ciertas plataformas.
- Los recursos del sistema distribuido pueden crecer en etapas, tomando ventaja de los últimos avances en tecnologías de redes y de computación.
- Las computadoras que conforman el sistema distribuido son en su mayoría estables, además no se necesita de ningún conocimiento especial para poder usarlas.
- Tolerancia a fallos en el desarrollo de aplicaciones.
- La computación distribuida facilita el trabajo colaborativo.

Todos estos factores se traducen en un tiempo reducido de desarrollo y depuración, y posiblemente en una implementación más efectiva de la aplicación.

2.3.4 Principales problemas de los sistemas distribuidos

En los sistemas distribuidos existen varios problemas fundamentales que deben ser resueltos; esto problemas están presentes por la

propia naturaleza de estos sistemas De hecho, algunos autores deciden incluso si un sistema es “más o menos distribuido” según el grado en que los presenta [REF.48]. Los más relevantes son los siguientes:

- **Fallo independiente.** Puede ocurrir que parte de los ordenadores que componen el sistema dejen de funcionar, y sin embargo el resto continúe funcionando correctamente. En este caso, por lo general se quiere que el sistema continúe prestando servicio, aunque quizás de forma reducida.
- **Comunicación no fiable.** Las redes que conectan los ordenadores no están, en general, exentas de errores. Es muy común que los datos transmitidos se pierdan, se dañen o se repitan por problemas en la red. El sistema distribuido ha de estar diseñado de forma que pueda controlar estas situaciones.
- **Comunicación insegura.** En general, los datos que se transmiten por la red pueden ser observados sin autorización, o incluso modificados intencionadamente.
- **Alto coste de la comunicación.** En la mayor parte de los casos la interconexión entre los ordenadores proporciona menos ancho de banda y más latencia que sus buses de comunicación internos.

2.3.5 Tendencias

Estaciones de trabajo que ejecutan millones de operaciones por segundo se están volviendo más comunes, y un continuo incremento en poder computacional es predicho. Cuando estas estaciones son interconectadas entre sí por medio de una red de alta velocidad, el poder computacional combinado puede ser utilizado para resolver problemas que requieran de una gran capacidad de procesamiento. De este modo la computación en red puede proveer un poder de procesamiento similar al de las supercomputadoras comerciales, y bajo ciertas circunstancias, el método basado en redes puede ser muy efectivo acoplando microprocesadores similares, obteniendo como resultado una configuración difícil de alcanzar, tanto económica como técnicamente hablando, por el hardware de una supercomputadora.

Para que la computación distribuida sea efectiva, la misma requiere de una alta velocidad de comunicación. En los últimos cincuenta años las velocidades de comunicación en red han ido incrementando considerablemente, véase la Fig. 2.3.5-1.

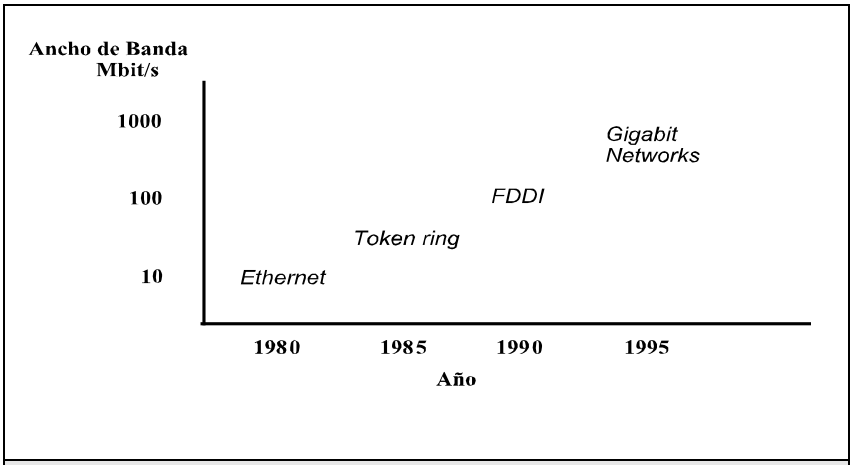


Figura 2.3.5-1. Velocidades de Red. [F9]

Entre los avances más notables en el desarrollo de la tecnología de redes tenemos los siguientes:

- Ethernet (IEEE 802.3)
- FDDI
- HiPPI
- SONET
- ATM

Estos avances obtenidos en el desarrollo de las tecnologías de redes de computadoras con alta capacidad de transferencia de datos y con baja latencia, hacen posible la implementación de lo que se hoy se denomina computación distribuida.

2.3.6 Comunicación en sistemas distribuidos

En los sistemas distribuidos, al no haber conexión física entre las distintas memorias de los equipos, la comunicación entre los procesos se la realiza mediante la transferencia de mensajes. Cuando un proceso A desea comunicarse con un proceso B, éste primero construye un mensaje en su propio espacio de direcciones; luego ejecuta una llamada al sistema que causa que el sistema operativo envié el mensaje al proceso B a través de la red.

Para el envío de mensajes se usa el estándar ISO OSI[1]. Las capas proporcionan varias interfaces con diferentes niveles de detalle, siendo la última capa la más general. El estándar OSI define las siguientes siete capas: física, enlace de datos, red, transporte, sesión, presentación y aplicación.

El modelo OSI distingue dos tipos de protocolos, los orientados hacia las conexiones y los protocolos sin conexión. En los primeros, antes de cualquier envío de datos se requiere una conexión virtual, que luego del envío deben finalizar. Los protocolos sin conexión no

[1] Un modelo por capas para la comunicación de sistemas abiertos.

requieren de este paso previo, y los mensajes se envían en forma de datagramas.

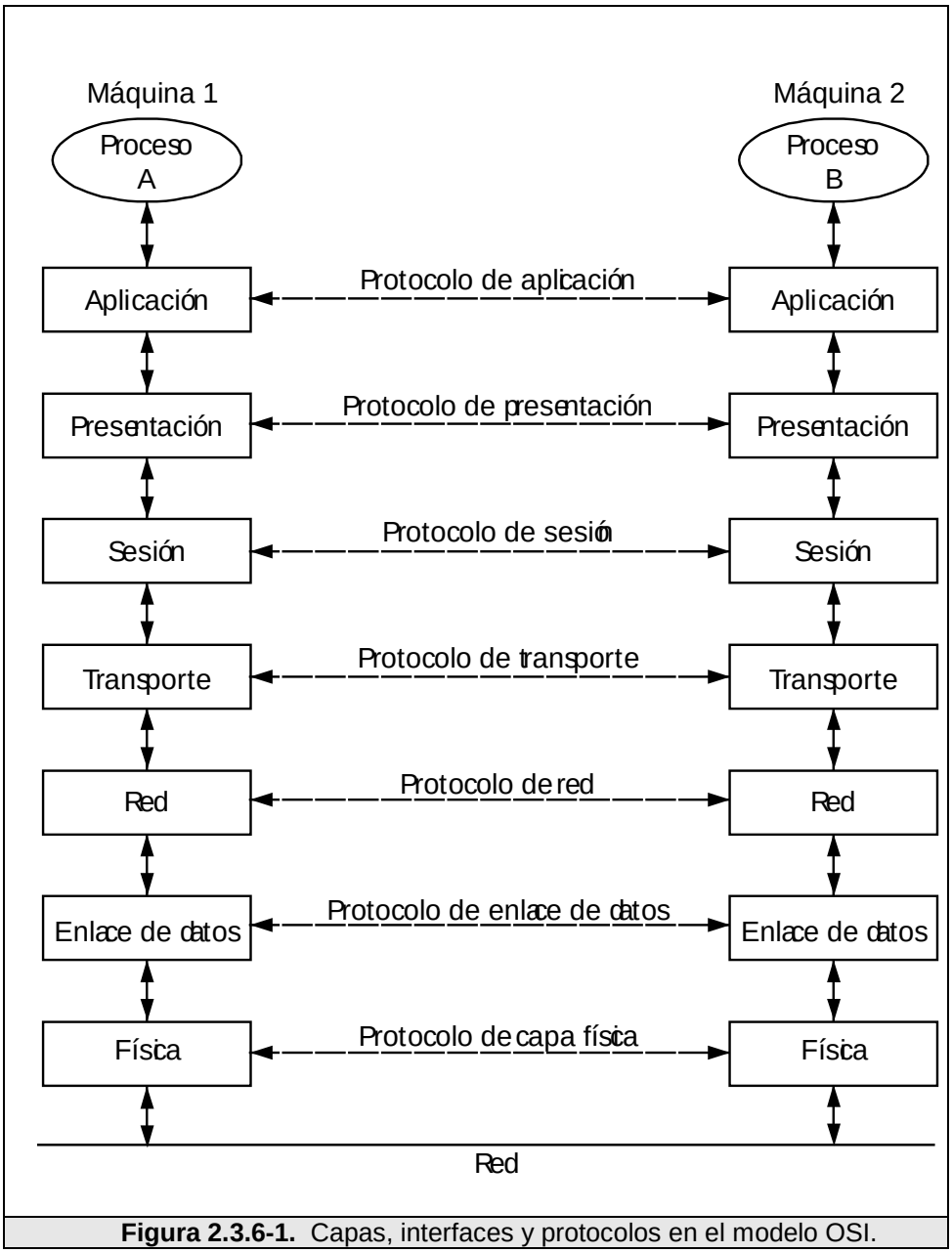


Figura 2.3.6-1. Capas, interfaces y protocolos en el modelo OSI.

2.3.6.1 Modo de transmisión asíncrona ATM

El modo de transmisión asíncrona o ATM[1] proporciona un rápido modo de transmisión. Las altas velocidades se alcanzan prescindiendo de la información de control de flujo y de control de errores en los nodos intermedios de la transmisión. ATM usa el modo orientado a conexión, y permite la transmisión de diferentes tipos de información, como voz, vídeo, datos, entre otros.

2.3.6.2 Modelo cliente – servidor

Los orígenes del modelo cliente-servidor se basan en los sistemas de paso de mensajes. La idea tras este modelo es estructurar el sistema operativo como un grupo de procesos cooperativos, llamados **servidores**, que ofrecen servicios a los usuarios, llamados **clientes**. Tanto las máquinas clientes, los servidores, por lo regular, ejecutan el mismo microkernel.

Este modelo basa la comunicación en una simplificación del modelo OSI. Las siete capas que proporciona producen un desaprovechamiento de la velocidad de transferencia de la red, con lo que sólo se usarán tres capas: física, enlace de datos y

[1] Asynchronous transfer mode networks.

solicitud/respuesta. Las transferencias se basan en el protocolo solicitud/respuesta y se elimina la necesidad de conexión.

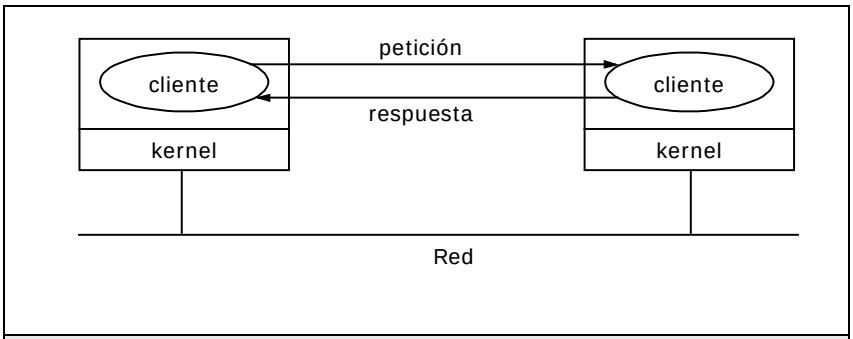


Figura 2.3.6.2-1. Modelo cliente-servidor.

Los datos, representados en forma de mensajes, se intercambian entre dos procesos, un emisor y un receptor. Un proceso envía un mensaje que representa una petición. El mensaje se entrega a un receptor, el cual procesa dicha petición y envía un mensaje como respuesta. En secuencia, la réplica puede disparar posteriores peticiones, que llevan a sucesivas respuestas, y así, sucesivamente.

Las operaciones básicas necesarias para dar soporte al modelo de paso de mensajes son **enviar** y **recibir**. Para comunicaciones orientadas a conexión, también se necesitan las operaciones **conectar** y **desconectar**.

El modelo cliente-servidor asigna roles diferentes a los dos procesos que colaboran. El **servidor** interpreta el papel de proveedor de servicio, esperando de forma pasiva la llegada de peticiones. El **cliente** invoca determinadas peticiones al servidor y aguarda sus respuestas. De una concepción simple, el modelo cliente-servidor proporciona una abstracción eficiente para facilitar servicios de red.

2.3.6.3 Modelo peer to peer (p2p)

En el paradigma peer to peer los procesos participantes cumplen los mismos papeles, con idénticas capacidades y responsabilidades, sugiriendo una interacción directa entre las partes. Cada participante puede solicitar una petición a cualquier otro y recibir una respuesta.

Mientras que el modelo cliente servidor es un modelo ideal para servicios centralizados de red, éste modelo resulta más apropiado para aplicaciones como transferencia de archivos, mensajería instantánea, video conferencia y trabajo colaborativo. Aunque también es posible que un sistema se base tanto en el modelo cliente servidor, como en el peer to peer.

Este modelo puede ser implementado en aplicaciones sencillas a través de bibliotecas de paso de mensajes, o utilizando tecnologías específicas en el caso del desarrollo de aplicaciones mucho más complejas.

2.3.6.4 Llamadas a procesos remotos (RPC)

Los RPC expanden el concepto de llamada local a procedimientos, y lo generalizan a una llamada a un procedimiento localizado en cualquier lugar de todo el sistema distribuido. En un sistema distribuido no se debería distinguir entre llamadas locales y RPCs, lo que favorece en gran medida la transparencia del sistema.

Una de las dificultades más evidentes a las que se enfrenta el RPC es el formato de parámetros en los procedimientos. Dicho problema se podría solucionar si ambos programas se ponen de acuerdo con el tipo de datos de los parámetros, o con el simple hecho de establecer un estándar en el formato de los parámetros, de forma que sea usado de forma única.

Otro problema más difícil de solucionar es el paso de punteros como parámetros. Debido a que los punteros guardan una dirección del espacio de direcciones local, el procedimiento que recibe el puntero

como parámetro no lo puede usar inmediatamente, ya que no tiene acceso a los datos, que para él son remotos.

Por último queda por solucionar la tolerancia a fallos. Una llamada a un procedimiento remoto puede fallar por motivos que antes no existían, como la pérdida de mensajes, o por el fallo del cliente o del servidor durante la ejecución del procedimiento.

2.3.7 La computación distribuida

La computación distribuida, no es otra cosa que el procesamiento que se lleva a cabo sobre un sistema distribuido. En la computación distribuida podemos distinguir lo siguiente:

- Servicio de red. Servicio proporcionado por un tipo de programa especial denominado servidor en una red.
- Aplicación de red. Aplicación para usuarios finales, que se ejecuta en computadoras conectadas a través de una red.

La diferencia entre servicios y aplicaciones de red no es siempre muy clara.

Como podemos deducir, la computación distribuida o procesamiento distribuido, supone no solamente la distribución de tareas entre

diferentes procesadores, como era el caso del procesamiento paralelo, sino más bien entre diferentes computadores, por lo que la comunicación entre las tareas se realiza a través de una red de área local [REF.23]. A cada computadora que forma parte del sistema distribuido se la denomina nodo.

La computación distribuida, algunas veces es denominada computación GRID[1], y se ha convertido en un modelo a utilizar para resolver problemas demasiado grandes para cualquier simple computadora o incluso supercomputadora. La manera de resolver dichos problemas es de proveer la potencia de computo de un grupo de computadoras en un a red. El problema es dividido en pequeñas partes, computacionalmente manejables, la cuales son procesadas por las computadoras que forman parte de la red. En otras palabras, todas las computadoras que forman parte de la red, colaboran entre sí para poder encontrar una solución a dicho problema en un tiempo relativamente corto.

Los proyectos de computación distribuida recientes han sido diseñados para utilizar las computadoras de centenares de millares

[1] Computación en grilla. Aunque algunas veces la computación en GRID en considerada un tipo de computación que evolucionó de la computación distribuida.

de voluntarios sobre todo el mundo, vía el Internet, para buscar señales de radio extraterrestres, buscar los números primos muy grandes con más de cien dígitos, y encontrar medicamentos más eficaces para luchar contra el virus del SIDA, por ejemplo. Estos proyectos tratan problemas muy grandes, y requieren tanto cálculo que las computadoras existentes hoy en día no servirían; o sería imposible encontrar una solución en un tiempo razonable utilizando el poder computacional de una computadora. Este modelo de computación distribuida, en donde se usan los recursos de computadoras voluntarias para resolver algún tipo problema, es denominado Computación Pública o Voluntaria [REF.49]. Este modelo se ha popularizado en los últimos años, proyectos como el ya mencionado SETI@home, hace uso de este tipo de computación distribuida, los procesos de cálculo poseen una prioridad muy baja dentro de la computadora y el cliente distribuido es ejecutado sólo mientras el procesador no está siendo usado, esto es usando los denominados “ciclos ociosos del procesador”[1].

Las ventajas de la computación distribuida son muy notorias. Además de poseer las mismas ventajas que nos ofrece los sistemas

[1] En muchas ocasiones se hace referencia a este tiempo por su definición en inglés: IDLE

distribuidos; podemos adicionar que la computación distribuida nos ofrece una manera sencilla y muy económica de crear una *computadora virtual* que posea una gran potencia de cálculo y un gran espacio de almacenamiento para ser utilizado en la resolución de problemas de gran envergadura, imposibles de resolver por una computadora normal.

2.3.7.1 Middleware

El middleware es el software de conectividad que está compuesto por un conjunto de servicios que permiten a varios procesos, que se ejecutan en una o varias máquinas, interactuar a través de la red. Es fundamental para migrar las aplicaciones monolíticas basadas en mainframes a aplicaciones cliente-servidor y proveer comunicación entre procesos a través de plataformas heterogéneas [REF.50].

Como vemos en la Fig. 2.2.7.1-1, el middleware es un conjunto software distribuido que se encuentra entre la aplicación y la plataforma concreta de red y sistema operativo, que posee el nodo sobre el cual se implementa la aplicación.

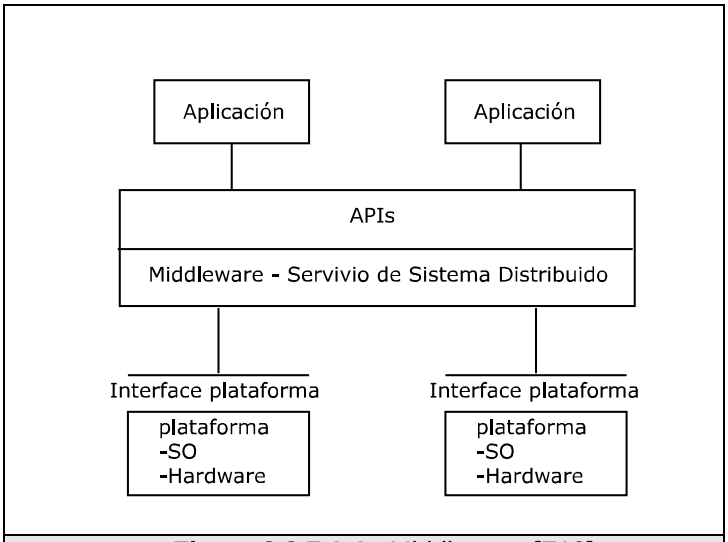


Figura 2.3.7.1-1. Middleware [F10].

Los servicios del Middleware proporcionan un conjunto de APIs[1] más funcional que el sistema operativo y los servicios de red para permitir a una aplicación:

- Localización transparente a través de la red, proveyendo interacción con otra aplicación o servicio.
- Ser independiente de los servicios de red.
- Ser fiable y disponible.
- Ser escalable, en el sentido de poder aumentar su capacidad sin pérdida de funcionalidad [REF.51].

[1] Interfaces de programación de aplicaciones, por su definición en ingles: Application Programming Interfaces

Con todo lo mencionado, podemos decir que el middleware tiene la finalidad de “virtualizar” los recursos de computación.

Entre las funciones principales del middleware, podemos mencionar:

- Encontrar un lugar adecuado para ejecutar las tareas solicitadas por el usuario.
- Optimizar el uso de recursos que pueden estar dispersos.
- Organizar el acceso eficiente a los datos.
- Autenticación de diferentes elementos.
- Se ocupa de las políticas de asignación de recursos.
- Ejecutar tareas.
- Monitorizar el progreso de los trabajos en ejecución.
- Gestionar la recuperación del sistema frente a fallos.
- Avisar del final de la tarea y devuelve resultados.

El middleware puede tomar una de las siguientes formas:

- **Monitores de procesamiento de transacciones o teleproceso**[1], los cuales proporcionan herramientas y un entorno para el desarrollo y explotación de aplicaciones distribuidas.

[1] Transaction Processing monitors (TP).

- **Llamadas a procedimientos remotos**[1], los cuales permiten que la lógica de una aplicación esté distribuida a través de una red.
- **Sistemas de Mensajes**[2], que proporciona intercambio de datos aplicación a aplicación, posibilitando la creación de aplicaciones distribuidas. Por ejemplo MPI o PVM.
- **Agentes de solicitud de objetos**[3], que permite que los objetos que componen una aplicación sean distribuidos y compartidos a través de redes heterogéneas.

---

[1] Remote Procedure Call (RPC).

[2] Message Oriented Middleware (MOM).

[3] Object Request Brokers (ORBs).

CAPÍTULO 3

3 ANÁLISIS

En el presente capítulo se analizan las posibles soluciones, tanto de hardware como de software, para resolver problemas que requieren gran capacidad de cómputo. Entre las soluciones de hardware que se tratan son el uso supercomputadoras, computadoras de alto rendimiento y clusters de computadoras heterogéneas.

Además se analizan las diferentes soluciones de middleware o software (PVM, MPI, BOINC) para la computación distribuida, poniendo énfasis en sus principales características, así como las ventajas y desventajas que estas poseen.

Por último, y en base al análisis elaborado, se describe la solución que se ha seleccionado en base a los recursos informáticos que posee la ESPOL.

3.1 Soluciones de hardware

El objetivo principal de la computación distribuida es unir el poder computacional de varios procesadores para resolver problemas de gran desafío computacional, en un tiempo relativamente corto.

Para lograr este poder computacional, tenemos las siguientes opciones: una computadora de alto rendimiento, una supercomputadora, o un cluster de computadoras.

3.1.1 Computadoras de alto rendimiento

La tecnología de computadoras de alto rendimiento (HPC[1]) es el uso de poder computacional como ayuda para la resolución de problemas altamente complejos de la manera más rápida y eficiente posible; esta tecnología es comúnmente utilizada en el campo científico y de ingeniería [REF.52].

Esto lo logra haciendo las operaciones secuenciales de manera más rápida y realizando procesos en paralelo [REF.53].

---

[1] Por sus siglas en inglés High Performance Computing.

Las computadoras de alto rendimiento están caracterizadas por:

- Cálculos rápidos.
- Memoria grande.
- Interconexión de alta velocidad.
- Entradas y salidas de alta velocidad [REF.53].

Los HPC son diseñados para sistemas de visualización, simulaciones y modelamiento de movimientos sísmicos, entre otros; pero sus costos son prohibitivos salvo para determinadas administraciones públicas e instituciones académicas. Son muchos los fabricantes actuales de este tipo de sistemas, siendo los más importantes SGI, Sun Microsystems e IBM.

La familia Silicon Graphics Prism (Fig. 3.1.1-1) fabricada por SGI, son computadoras de alto rendimiento diseñadas para sistemas de visualización. La computadora más avanzada que pertenece a esta familia, posee hasta 256 procesadores Intel Itanium II, más de 3 terabytes de memoria; y tiene un costo superior a los 2 millones de dólares. [REF.54].

Figura 3.1.1-1. Silicon Graphics Prism [F18]

El Sun Fire V40z Server (figura 3.1.1-2) de Sun Microsystems, diseñado principalmente para sistemas financieros inteligentes tales como los sistemas ERP/CRM, virtualización y consolidación de servidores[1], y granjas de cómputo para el modelamiento y simulación; tiene un costo aproximado de 26.000 dólares. Sus características principales son:

- 4 Procesadores AMD Opteron Modelo 880 (Doble Núcleo).
- 16 GB de Memoria.
- 2 Discos SCSI Ultra 320 10K RPM [REF.55].

[1] Involucra la emulación de multiples servidores en una plataforma de Hardware.

Figura 3.1.1-2. Sun Fire V40z Server [F19]

3.1.2 Supercomputadoras

Este tipo de computadoras son utilizadas generalmente para procesar gran cantidad de datos; simulaciones científicas, animaciones gráficas, análisis estructurales, diseño electrónico, meteorología, son algunos de los campos en los cuales son usadas. Uno de los fabricantes de supercomputadoras más conocido es la empresa *Cray Inc*. [REF.56]

Por su alto costo, tienen un uso limitado a organismos gubernamentales, militares y grandes centros de investigación, para uso en aplicaciones científicas, como en la simulación de procesos naturales (predicción del tiempo, análisis de cambios climáticos,

entre otros procesos), modelamiento molecular, simulaciones físicas como túneles de viento, criptoanálisis, etc.[REF.57].

La BLUE GENE/L de IBM (fig 3.1.2-1), es hoy en día la supercomputadora más poderosa del mundo según el sitio Web www.top500.org. Cuenta con 131.072 procesadores PowerPC, 32.768 GB de memoria y con una velocidad que alcanza los 280.600 gigaflops. Su construcción tuvo un costo aproximado de 150 millones de dólares [REF.58].

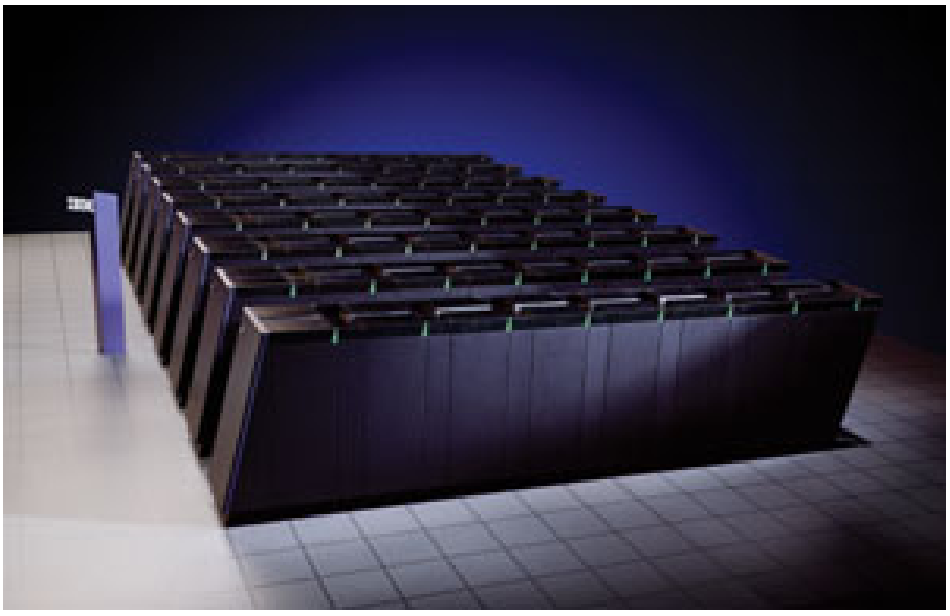


Figura 3.1.2-1. IBM BLUE GENE/L [F12]

Como podemos observar, a pesar de que una supercomputadora nos puede brindar un poder de procesamiento extremadamente alto; sus

costos son demasiado elevados. Otro ejemplo es SGI Altix 3700, creado por la NASA como sistema de soporte en las misiones de despegue; el cual cuenta con un total de 10.240 procesadores Intel Itanium II, 20 terabytes de memoria y una velocidad que llega a los 6 gigaflops; el costo de construcción fue de aproximadamente 160 millones de dólares [REF.59].

3.1.3 Clusters de computadoras

Los clusters ó racimo de computadoras consisten de un grupo de computadoras independientes de relativo bajo costo, cada una de las cuales tiene su propio sistema operativo (generalmente Linux); conectadas entre sí mediante un sistema de red de alta velocidad (gigabit de fibra óptica por lo general). Este grupo de computadoras forma un sistema potente que aparece ante los clientes y aplicaciones como una sola computadora de gran poder de procesamiento [REF.34].

Entre la principal ventaja del uso de clusters tenemos que las computadoras que lo conforman pueden tener todas las mismas configuraciones de hardware y sistema operativo (cluster homogéneo), diferente rendimiento pero con arquitecturas y sistemas operativos similares (cluster semi-heterogéneo) o tener diferente

hardware y sistema operativo (cluster heterogéneo), lo que hace más fácil y económica su construcción [REF.60].

Según su funcionalidad, existen tres tipos de cluster: Los *clusters de alto rendimiento*, en donde lo que se busca es un gran poder de procesamiento, este tipo de clusters se aplica mejor en problemas grandes y complejos que requieren una cantidad enorme de potencia computacional; *los clusters de servidores virtuales*, destinado al balanceo de carga, permite que un conjunto de servidores de red compartan la carga de trabajo y de tráfico de sus clientes, aunque aparezcan para estos clientes como un único servidor, y por último los *clusters de alta disponibilidad*, trata del mantenimiento de servidores que actúen entre ellos como respaldos de la información que sirven, también conocidos como clusters de redundancia. Entre todos estos, los clusters de alto rendimiento se ajustan más a los objetivos que se buscan.

Tenemos por ejemplo, el grupo de investigación Genetic Programming en alianza con el Centro de Ensamblaje de COMPAQ en Sunnyvale, construyó un cluster de alto rendimiento compuesto de 1000 computadoras Pentium II de 350 MHz (Fig 3.1.3-1), destinado a la investigación de algoritmos genéticos. Este cluster tiene un total de

64 gigabytes de memoria, y un poder de procesamiento aproximado de 350 gigaflops. Su construcción duró casi 4 años y tuvo un costo aproximado de 1,2 millones de dólares [REF.61].

Figura 3.1.3-1. 1000-Pentium Beowulf-Style Cluster Computer [F13]

Los clusters de computadoras pueden ser utilizados como una alternativa de un relativo bajo costo, para procesamiento en paralelo de aplicaciones científicas. Por ejemplo el *Avalon*, un *cluster* de 140

computadores que se encuentra entre las 500 máquinas más potentes del planeta; puede alcanzar una velocidad de 47.7 gigaflops, y fue construido utilizando un presupuesto de sólo 340,000 dólares [REF.62].

Las principales desventajas de los clusters es que no existe mucho software capaz de tratar al cluster como un único sistema. Además, las redes de computadoras no están diseñadas para el procesamiento en paralelo, por lo que muchas veces el ancho de banda es demasiado bajo y su latencia demasiado alta en comparación con los buses de datos que utilizan las supercomputadoras [REF.63]. Esta última puede ser superada, mediante el uso de otras alternativas al Ethernet, tales como InfiniBand.

InfiniBand es una arquitectura que permite la conexión de redes a alta velocidad. Mientras que Ethernet alcanza 1 gigabit por segundo, la versión 4x de InfiniBand puede alcanzar hasta 10 gigabits por segundo, y en la actualidad se está trabajando en una versión 12x, que triplicaría esta velocidad. El problema con esta tecnología es que todavía es costosa en comparación con Ethernet, por ejemplo, un

juego de un *switch* y tarjetas para conectar cuatro computadoras tiene en conjunto un costo aproximado de 12 mil dólares [REF.64].

Existen varios métodos para la construcción de un cluster de alto rendimiento, entre las cuales tenemos Beowulf y Oscar.

Beowulf es una tecnología para agrupar computadoras basadas en el sistema operativo Linux, para formar una supercomputadora virtual paralela; no es un paquete de software especial, ni una nueva topología de red, ni un núcleo de sistema operativo modificado; consiste de un nodo maestro y uno o más nodos esclavos conectados en red, construido con elementos de hardware comunes en el mercado, similar a cualquier PC capaz de ejecutar Linux, adaptadores de red y switches estándares [REF.34]. Los nodos en el cluster están dedicados exclusivamente a ejecutar tareas asignadas al cluster.

Entre las ventajas de usar cluster Beowoulf tenemos:

- Tiempo relativamente corto de construcción.
- Bajo costo por cada nodo.
- Extremadamente fácil y barato para expandirse.

- El administrador responsable del diseño del sistema tiene el control sobre qué componentes serán usados, especificaciones de sistema operativo, entre otros.
- Fácil de actualizar con la tecnología actual.
- Hace posible el uso de diferentes arquitecturas de hardware.
- Muy estable y robusto.
- Fácil de reorganizar la topología de los nodos para ajustarse a su uso [REF.65].

Las desventajas del uso de Beowulf son las siguientes:

- La comunicación puede ser baja – Un sistema Beowulf se es considerado generalmente como una solución óptima para trabajos altamente paralelos, donde la comunicación es mínima
- La responsabilidad del diseño y su administración toma tiempo y energía aparte del trabajo para la aplicación.
- No existe un soporte oficial, es decir el usuario debe guiarse por información publicada en el Internet o grupos de noticias para la ayuda con problemas que puedan surgir [REF.65].

Por otro lado, Open Source Cluster Application Resources (OSCAR) es una recopilación de los mejores métodos conocidos para construir, programar y usar cluster de alto rendimiento. Consiste de

un paquete de software que contiene todos los programas necesarios para instalar, construir, mantener y utilizar un cluster sobre Linux de tamaño modesto [REF.66].

La arquitectura de un cluster OSCAR es muy similar a la de de un cluster Beowulf. Se configura un nodo como maestro, el cual provee los servicios a los nodos de cómputo. En el cluster Beowulf, una vez que se ha configurado el nodo maestro, se deben construir cada uno de los nodos de cálculo. OSCAR, en cambio, asiste en la configuración del nodo principal y construye los nodos de cálculo basados en una descripción especificada por el usuario. Esta construcción y configuración reduce considerablemente el esfuerzo y la necesidad de habilidades especiales para conformar un cluster [REF.67].

3.1.3.1 Computación de ciclos redundantes

También llamado NOWs[1], un tipo especial de cluster de alto rendimiento, el modelo de computación de ciclos redundantes, también conocido como computación zombi, consiste en que un servidor o grupo de servidores distribuyen trabajo de procesamiento

[1] Por sus siglas en ingles Network of Workstations (Redes de estaciones de trabajo).

a un grupo de computadoras voluntarias con la finalidad de ceder capacidad de procesamiento no utilizada. Básicamente, cuando se deja la computadora encendida, pero sin utilizarla, la capacidad de procesamiento se desperdicia por lo general en algún protector de pantalla, este tipo de procesamiento distribuido utiliza el poder computacional de la computadora cuando no se la necesita, aprovechando al máximo su capacidad de procesamiento.[REF.57]

Entre las principales ventajas de utilizar esta tecnología tenemos:

- Ahorro: no se debe incurrir en costos de adquisición de nodos de trabajo, ya que se usan estaciones de trabajo ya existentes, que a su vez son usadas para otros propósitos.
- Mantenimiento y disponibilidad: dado que los elementos que forman un cluster se encuentran fácilmente en el mercado (son componentes de producción masiva y por lo tanto de bajo costo), al fallar alguno de ellos se puede reemplazar sin mayores inconvenientes.
- Hospedaje: a diferencia de las supercomputadoras, este tipo de sistemas no requiere de un sitio de alojamiento especial, debido a que en la realidad se crea una supercomputadora virtual formada por el poder de cómputo de todas las estaciones de trabajo que forman el sistema.

- Alta escalabilidad: al ser público, se podría contar siempre con computadoras que cedan voluntariamente su tiempo de ocio en la ejecución de una aplicación, el cual se incrementa con el paso del tiempo, tal como lo realiza actualmente el proyecto SETI@HOME. [REF.57]

Entre las desventajas podemos anotar al utilizar computación de ciclos redundantes, podemos anotar las siguientes:

- Poder computacional muy variable; ya que los nodos no están dedicados exclusivamente a procesar datos para el cluster, sino también a aplicaciones propias de la estación de trabajo, el poder computacional total del sistema será siempre variable.
- Latencia impredecible en la comunicación; como la red de interconexión no está aislada del resto de la red, ésta es afectada por el tráfico generado por el resto de aplicaciones que utilizan la red.
- Seguridad y confiabilidad de los resultados: ya que los datos pueden ser transmitidos a través de una red pública, y ser procesados por nodos ajenos a la organización; los resultados obtenidos no son 100% confiables.

La idea de utilizar los ciclos redundantes del CPU para el procesamiento de datos, fue originalmente utilizada por SETI para procesar las señales de radio provenientes del espacio en búsqueda de patrones inteligentes, proyecto auspiciado por la NASA.

En el año 2001 la Universidad de Berkeley en California, inicia el proyecto BOINC, una plataforma de software para el desarrollo de aplicaciones de computación distribuida. Hoy en día, BOINC da soporte no sólo al proyecto SETI, sino a una gran variedad de proyectos científicos que necesitan de una gran cantidad de poder computacional.

BOINC es un conjunto de aplicaciones, herramientas y APIs de programación que permiten la creación y administración de un cluster de cómputo basado en ciclos redundantes, de manera fácil y rápida.

3.2 Middleware para computación distribuida

Hoy en día existe una gran cantidad de software para computación distribuida, entre los más importantes tenemos: Máquina Virtual Paralela (PVM), Interfase de paso de mensajes (MPI) y Infraestructura abierta de Berkeley para la computación a través de redes (BOINC).

3.2.1 Máquina Virtual Paralela (PVM)

La Máquina Virtual Paralela (PVM) fue desarrollada en el año 1989 por el Oak Ridge National Laboratory en conjunto con otros centros de investigación. PVM, es un conjunto de herramientas de software y librerías para crear y ejecutar aplicaciones concurrentes o paralelas, basado en el modelo de paso de mensajes.

Este conjunto librerías posibilita a un grupo de computadoras heterogéneas, con sistema operativo Linux, conectadas en red, ser vistas como una sola máquina virtual paralela con una gran capacidad de cómputo. PVM, maneja de forma transparente todo el paso de mensajes, conversión de datos, y programación de tareas [REF.68].

PVM funciona eficientemente en la mayoría de sistemas distribuidos, así como también en sistemas de memoria compartida y procesadores masivamente paralelos (MPPs). En PVM, los usuarios descomponen la aplicación en tareas separadas y escriben sus aplicaciones como una colección de tareas colaborativas. Una aplicación PVM, se ejecuta en una maquina virtual creada por el ambiente PVM, el cual inicia y termina las tareas además de proveer

los servicios de comunicación y sincronización entre las tareas [REF.68].

Las primitivas del paso de mensajes de PVM son orientadas hacia las operaciones heterogéneas, involucrando las estructuras fuertemente definidas para el buffering y transmisión. Las estructuras de comunicación incluyen aquellas que sirven para enviar y recibir estructuras de datos, así como también primitivas de alto nivel tales como broadcast, sincronización de barrera, y suma global. Las comunicaciones intraprocesos en PVM pueden ser realizadas ya sea mediante el uso de paso de mensajes o memoria compartida. Para dar soporte a las primitivas de memoria compartida, PVM debe emular un modelo de memoria compartida usando las primitivas de paso de mensaje de PVM, lo cual conlleva una gran cabecera de sus primitivas DSM (Distributed Shared Memory). PVM soporta las operaciones de comunicación en grupo tales como creación de grupos dinámicos, unión a grupos, y abandono de grupo. PVM es ampliamente usado en ambientes de computación heterogénea distribuida debido a su eficiencia en el manejo de la heterogeneidad, escalabilidad, y balanceo de carga.[REF.68]

Los mensajes pasados entre los nodos usan el protocolo de comunicación TCP/IP, además del programa rsh[1] para iniciar las sesiones entra las máquinas. Este comando (rsh) permite correr comandos de Linux de manera remota.

En general, el sistema PVM está compuesto de dos partes:

- **El demonio PVM;** llamado *pvmd3*, algunas veces llamado solamente *pvmd*, que reside en cada una de las computadoras configurada en la máquina virtual paralela. Es encargado de controlar el funcionamiento de los procesos de usuario en la aplicación PVM y de coordinar las comunicaciones. El demonio pvmd3, es diseñado de tal forma que cualquier usuario pueda instalarlo. Cuando un usuario quiere ejecutar una aplicación PVM, primero debe crear una máquina virtual iniciando PVM. De esta forma, la aplicación PVM puede ser iniciada por consola desde cualquier computadora nodo.
- **La librería de rutinas de interfase PVM;** que contiene un repertorio completo de las primitivas que son necesarias para la cooperación entre las tareas de una aplicación. Esta librería contiene rutinas de paso de mensaje que pueden ser llamadas

[1] Una utilidad de Linux que permite iniciar una sesiión de manera remota, o ejecutar un commando en una máquina remota.

por los usuarios, generar procesos, iniciar y terminar tareas, coordinación de tareas y modificación de la máquina virtual [REF.69].

Este modelo de computación se basa en la idea de que una aplicación consiste de varias tareas. Cada tarea es responsable por una parte de trabajo computacional. Algunas veces una aplicación es paralelizada en base a sus funciones, esto es, cada tarea ejecuta una función diferente, por ejemplo configuración, solución, entradas y salidas, y presentación. Como mencionamos en la sección 2.2.3.5, este proceso es también llamado paralelismo de control. Otro método más común de paralelización de una aplicación es llamado paralelismo de datos, véase sección 2.2.3.3. PVM suporta cada uno de esto de modelos, o una mezcla de ellos. Dependiendo de su funcionalidad, las tareas pueden ejecutarse en paralelo, y muchas de ellas pueden necesitar sincronización o intercambio de datos, aunque este no siempre sea el caso. Un diagrama explicativo del modelo de computación PVM es mostrado en la Fig. 3.2.1-1.[REF.69]

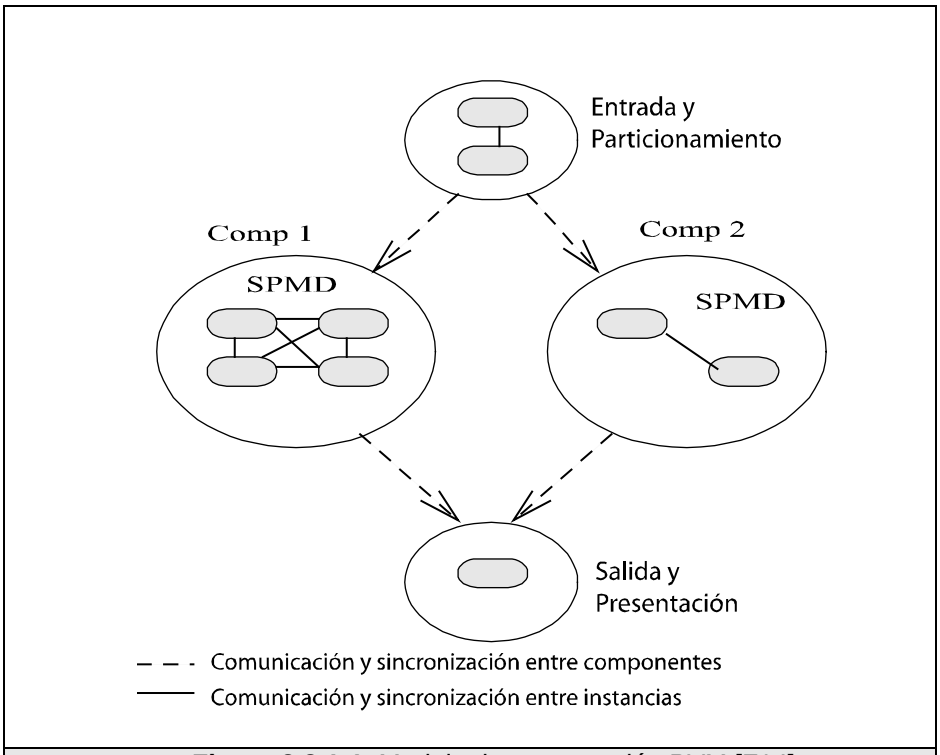


Figura 3.2.1-1. Modelo de computación PVM [F14]

En cuanto a la programación, PVM trabaja de la siguiente forma:

- El programador escribe un programa maestro, y uno o más programas esclavos en C o Fortran 77 haciendo uso de las librerías PVM. El programa maestro corresponde a la tarea maestra y los programas esclavos corresponden a un grupo de tareas esclavas.
- La tarea maestro es iniciada; lo que ocasiona que las tareas esclavas se inicien de manera concurrente con ésta, a través de llamadas PVM.
- Las tareas esclavas, en algún momento pueden convertirse en maestras de nuevas tareas esclavas.

Entre las ventajas del uso de PVM podemos mencionar las siguientes:

- Fácil instalación y uso, con una interfase de programación simple y completa.
- Es un software de dominio público con creciente aceptación y uso.
- Flexible, ya que se adapta a diversas arquitecturas, permite combinaciones de redes locales con las de área extendida, cada aplicación "decide" donde y cuando sus componentes son ejecutados y determina su propio control y dependencia, se pueden programar diferentes componentes en diferentes lenguajes, es fácil la definición y la posterior modificación de la propia Máquina Virtual Paralela.
- Permite aprovechar el hardware existente.
- Explota la heterogeneidad natural de las computadoras.
- Existe una optimización del rendimiento gracias a la asignación de tareas a la arquitectura más apropiada.
- Además de ser portable, expande esta definición de portabilidad incluyendo interoperabilidad, permitiendo la comunicación entre sistemas de diferente arquitectura.

- Maneja eventos para facilitar la implementación de la tolerancia a fallos, esto es, provee de un esquema de tolerancia a fallos.

Por otro lado, PVM tiene las siguientes desventajas:

- Sólo se permite el paso de tipos de datos primitivos, es decir si se desea transmitir un objeto entre componentes distribuidos, se deben tomar consideraciones especiales (serialización y deserialización de objetos).
- El programador tiene que implementar los detalles de paralelización de forma explícita.
- Las aplicaciones PVM son difíciles de depurar [REF.70].
- Puede llegar a ser inseguro en un ambiente de redes públicas como el Internet.

3.2.2 Interfase de Paso de Mensajes (MPI)

A diferencia de otras herramientas de paso de mensajes, la primera versión de MPI fue completada en Abril de 1994 por un consorcio de más de 40 miembros asesores de computación distribuida y paralela de alto rendimiento. Este esfuerzo tuvo como resultado la definición tanto de la sintaxis y semántica de un conjunto de librerías de paso de mensajes en un ambiente con memoria distribuida.

En el modelo de programación MPI, un computo comprende de uno o más procesos comunicados a través de llamadas a rutinas de librerías para mandar (`send`) y recibir (`receive`) mensajes a otros procesos. En la mayoría de las implementaciones de MPI, se crea un conjunto fijo de procesos al inicializar el programa, y un proceso es creado por cada tarea. Sin embargo, estos procesos pueden ejecutar diferentes programas. De ahí que, el modelo de programación que subyace tras MPI es MIMD, aunque se dan especiales facilidades para la utilización del modelo SIMD. Es decir, MPI fue desarrollado tanto para dar soporte a los grandes computadores de memoria compartida (como los MPPs), así como también para clusters o redes de ordenadores.

Las principales ventajas de establecer un estándar para el paso de mensajes son la portabilidad y facilidad de uso. En un ambiente de memoria distribuida en la cual las rutinas de más alto nivel y/o abstracciones son construidas en base a las rutinas de paso de mensajes en más bajo nivel, los beneficios de estandarización son particularmente aparentes [REF.68].

Las principales características de MPI son:

- *Servicios de comunicación*. MPI tiene un gran conjunto de servicios de comunicación colectiva y de comunicación punto a punto. Además, proporciona operaciones para la creación y administración de grupos en una forma escalable.
- *Comunicaciones totalmente asíncronas*.
- *Tipos de datos definidos por el usuario*. MPI tiene un mecanismo extremadamente poderoso y flexible para describir rutinas de movimiento de datos, tanto para tipos de datos predefinidos como derivados [REF.68].
- *Gran soporte para sistemas MPP y clusters*. Una topología virtual que refleja el patrón de comunicación de la aplicación puede ser asociada con un grupo de procesos. MPI proporciona una abstracción de alto nivel para la topología de paso de mensajes, de modo que las topologías de aplicaciones en general son especificadas por un grafo, y cada proceso de comunicación es representado por un arco [REF.68].

Los elementos básicos de MPI son una definición de un interfaz de programación independiente del lenguaje, en adición de una colección de bindings o concreciones de esa interfaz para los lenguajes de programación más extendidos en la comunidad de computadores paralelos: C y FORTRAN.

Un programador que quiera emplear MPI para sus proyectos trabajará con una implementación concreta de MPI, que constará de, al menos, estos elementos:

- Una biblioteca de funciones para C, y una colección de constantes y macros.
- Una biblioteca de funciones para FORTRAN.
- Comandos para la compilación y ejecución de aplicaciones paralelas.
- Herramientas para monitorización y depuración.
- Provee mecanismos para el manejo de la tolerancia de fallos.

Al ejecutar una aplicación se lanzan en paralelo N copias del mismo programa (procesos). Estos procesos no avanzan sincronizados instrucción a instrucción sino que la sincronización, cuando sea necesaria, tiene que ser explícita. Los procesos tienen un espacio de memoria completamente separado. El intercambio de información, así como la sincronización, se hacen mediante paso de mensajes.

Entre las ventajas de MPI podemos mencionar las siguientes:

- Basado en una librería, funciona con muchos compiladores.
- Existen versiones de libre distribución

- MPI está estandarizado en muchos niveles. Con una sintaxis estandarizada, el código MPI puede ser ejecutado bajo cualquier implementación MPI ejecutándose sobre un sistema.
- Existen versiones optimizadas.
- Provee herramientas para el control y depuración.
- Es portable.

Por otro lado, MPI tiene las siguientes desventajas:

- Comunicaciones deben programarse explícitamente.
- Las implementaciones no siempre son óptimas en modelos de memoria compartida.
- Aunque el usuario pueda usar varias implementaciones de MPI en un sistema, el rendimiento puede variar en diferentes implementaciones.
- A diferencia de PVM, los nodos de cómputo no pueden ser agregados o quitados de manera dinámica, solo de manera estática.
- Nada en el estándar MPI describe la cooperación entre redes y arquitecturas heterogéneas, en otras palabras no es interoperable [REF.71].

3.2.3 Infraestructura abierta de Berkeley para la computación a través de redes (BOINC)

BOINC es una plataforma middleware que permite a los científicos hacer supercomputación usando los ciclos ociosos en las computadoras del público en general. Esta es una tendencia a la cual se la denomina “computación voluntaria”. La computación voluntaria dio sus inicios en mediados de 1990 con dos proyectos revolucionarios: GIMPS [1] y distributed.net [REF.72].

La computación voluntaria requiere de la siguiente infraestructura de software: un servidor que se encargue de distribuir el trabajo y mantenga el rastro de los participantes; y un cliente que se encargue de la comunicación, administración del procesamiento y almacenamiento, y muestre información gráfica [REF.72].

La principal razón por la cual desarrolló BOINC fue por el interés de poder compartir con la comunidad científica, aquel poder computacional, que muchas veces le resultaba excesivo para la

[1] Por sus siglas en inglés Great Internet Mersenne Prime Search (Gran Búsqueda en Internet del Número Primo Mersenne)

cantidad de datos que tenían que procesar, a partir de esto se creó BOINC.

El sistema BOINC consiste de las siguientes partes: un **cliente** el cual es compartido entre todos los proyectos de BOINC, un **grupo de servidores** para la administración de proyectos y aplicaciones distribuidas [REF.74].

El grupo de servidores se encarga de dividir un problema en un conjunto de tareas que deben ser procesadas por los clientes. Estas tareas son enviadas a cada uno de los clientes para ser procesada y enviar los resultados de regreso para ser analizados. BOINC tiene la característica de ser portable, en el sentido de que el cliente puede ser instalado en la mayoría de plataformas existentes, entre ellas Linux, Windows y Macintosh. La comunicación que se realiza entre el grupo de servidores y los clientes es a través de mensajes que consisten de archivos XML, sobre el protocolo HTTP.

BOINC maneja la metodología de proyectos y aplicaciones distribuidas. Donde un proyecto es un grupo de una o más aplicaciones distribuidas. Loa clientes de BOINC pueden participar

en el procesamiento de datos de uno o más proyectos al mismo tiempo.

Una de las características principales de BOINC es su capacidad para administrar de manera sencilla aplicaciones distribuidas. La aplicación, la cual es capaz de procesar una tarea y generar un resultado, es enviada al cliente, junto con las tareas que debe realizar.

Entre las principales ventajas de BOINC tenemos las siguientes:

- Entorno flexible para las aplicaciones; las aplicaciones existentes en lenguajes comunes (C, C++, Fortran) pueden funcionar como aplicaciones BOINC con poco o ninguna modificación; incluso una aplicación puede ser un grupo de varios archivos (programas múltiples y un script de coordinación, por ejemplo). Además las nuevas versiones de una aplicación se pueden distribuir sin intervención del usuario.
- Seguridad; BOINC protege los proyectos contra varios tipos de ataques. Por ejemplo, utiliza firmas digitales basadas en encriptación de claves públicas para protegerlas de la distribución de virus.

- Servidores múltiples y tolerancia a fallos; los proyectos pueden contar con servidores redundantes. Los clientes automáticamente intentan servidores alternativos; si todos los servidores están fuera de funcionamiento, los clientes hacen reintentos exponenciales en el tiempo para evitar saturar los servidores cuando estén en marcha de nuevo.
- Provee de herramientas de seguimiento de sistema; BOINC incluye un sistema de administración basado en Web, para poder mostrar las variaciones en el tiempo (carga de CPU, tráfico de red, tamaños de las tablas de la base de datos). Esto simplifica la tarea de diagnosticar problemas de rendimiento.
- Es código abierto; BOINC es un proyecto de Código Abierto (Open source), sin embargo los programas de las aplicaciones no necesitan estar en código abierto. BOINC puede utilizarse para proyectos de computación ya sean privados, públicos o comerciales. Cada proyecto debe poseer y mantener sus propios sistemas servidores. Estos sistemas se pueden instalar facilmente usando componentes en Código Abierto (MySQL, PHP, Apache).
- Provee soporte para grandes bloques de datos. BOINC permite aplicaciones que producen o consumen grandes cantidades de datos, o que usan gran cantidad de memoria. La distribución de datos y la recolección se pueden esparcir en muchos servidores,

y las computadoras cliente transferir muchos datos sin obstruirse. Los usuarios pueden especificar límites en el uso de disco y en el ancho de banda. El trabajo sólo se envía a computadoras capaces de realizarlo.

- Plataformas de usuarios múltiples; el programa cliente de BOINC está disponible en las plataformas más habituales (Mac OS X, Windows, Linux y otros sistemas Unix). El cliente puede usar múltiples procesadores.
- Provee de una interfase Web para los participantes; BOINC facilita el entorno Web para la creación de cuentas, edición de preferencias, y estado de cuentas. Las preferencias de un usuario son inmediatamente propagadas a todos sus computadores, haciendo fácil manejar gran cantidad de computadoras.
- Provee de un sistema de almacenamiento de unidades configurable; el programa cliente descarga suficiente trabajo para mantener ocupado la computadora durante la cantidad de tiempo especificada por el usuario. Esto se puede hacer para disminuir la frecuencia de las conexiones al servidor o para permitir al computador seguir funcionando aunque el servidor del proyecto no esté operativo [REF.73].

Entre las desventajas de BOINC tenemos:

- No da soporte para la interoperabilidad de aplicaciones, es decir una aplicación que se ejecuta en un cliente no se puede comunicar con otra que se este ejecutando en otro cliente.
- La portabilidad radica en el sentido de que el sistema envía la aplicación distribuida al cliente dependiendo de su arquitectura, de tal forma que se debe desarrollar versiones de la misma aplicación para diferentes arquitecturas.
- Orientado sólo a la paralelización de datos.
- Usar XML sobre HTTP para la comunicación puede dar como resultado paquetes de comunicación muy grandes.

3.3 Estudio de las alternativas

A continuación se procede a realizar una revisión de las principales características del proyecto a ser implementado, estableciendo los requerimientos necesarios para su posterior implementación. Luego de establecer los requerimientos, se procede al análisis y elección de las alternativas de hardware y software adecuadas para su puesta en marcha.

3.3.1 Requerimientos del sistema

El principal requerimiento es que el cluster a ser desarrollado e implementado sea de bajo presupuesto de inversión sin comprometer su rendimiento computacional, en el cual pueda aprovechar la capacidad de procesamiento de computadoras ya existentes en la ESPOL. Los nodos serán computadoras de escritorio utilizadas para labores diarias dentro de la institución, esto implica que no estarán dedicadas exclusivamente a procesar datos para el cluster.

En general, el sistema deberá contar con las siguientes características y funcionalidades:

- Proveer mecanismos que permitan agregar nodos de una manera ágil y sencilla; y ser fácilmente escalable.
- Ser adaptable en el desarrollo de diferentes tipos de aplicaciones distribuidas.
- Dividir el trabajo en tareas más pequeñas, y distribuirlas a los nodos clientes del sistema para ser procesadas.
- Ser independiente de la plataforma.
- Ser tolerante a fallos.
- Ser seguro y estar protegido de ataques.
- Ser fácilmente configurable para que cada cliente procese trabajo en el tiempo y situación que el sistema crea conveniente, en

nuestro caso cuando el usuario no esté usando su computadora, aprovechando de esta forma los ciclos ociosos del CPU.

- Deberá medir el poder de procesamiento de cada cliente
- Proveer de herramientas que faciliten la creación de aplicaciones distribuidas.

3.3.2 Análisis de alternativas y selección de la solución más apropiada

En el análisis de las alternativas de hardware claramente se indica que los supercomputadores son las máquinas con mayor poder de procesamiento que cualquier otra pero el costo elevado en su adquisición es un gran limitante. Por otro lado, aunque las computadoras de alto rendimiento tengan un costo mucho menor que las supercomputadoras, aun sigue siendo elevado. Sin embargo, la alternativa de clusters de computadoras es la más viable, por cuanto permite construir un sistema distribuido de gran capacidad de procesamiento, incluso superior a la de un supercomputador; fácilmente escalable, tolerante a fallos, y demás ventajas que nos ofrecen los clusters, con relativa poca inversión. Este cluster puede ser construido con computadores que se estén siendo dados de baja o con los que actualmente cuenta la universidad y se deseen agregar al sistema distribuido de forma voluntaria.

Luego de haber concluido que la mejor opción es la de construir un cluster de computadoras, se debe definir la plataforma o middleware necesaria para la construcción del cluster la cual cumpla con los requerimientos planteados en la sección anterior. En la sección 3.2 se analizaron tres diferentes tipos de alternativas disponibles.

La primera opción (PVM) es la pionera en sistemas distribuidos pero tiene que ser descartada porque a pesar de ser totalmente gratuita y de tener la capacidad de explotar la heterogeneidad de los sistemas, se debe de manejar de forma explicita los detalles de paralelización, lo que hace que el desarrollo de aplicaciones distribuidas pueda tomar mucho tiempo y esfuerzo. Además PVM no brinda la facilidad para configurar los clientes de tal forma que procesen tareas usando los ciclos ociosos del CPU.

La segunda opción (MPI), es una alternativa similar a PVM, con algunas mejoras en cuanto a la estandarización y manejo de funciones; al contrario de PVM, MPI está más orientada a ser utilizada sobre sistemas MPPs, es por esto que su uso en sistemas de tipo cluster no es muy maduro. Además tenemos que tanto MPI, así como PVM, no brindan un nivel de seguridad adecuado para su

funcionamiento en redes de acceso público, y para contrarrestar esta desventaja en ambos casos, implicaría realizar una inversión adicional para adquirir mejor tecnología de seguridad en redes.

Finalmente tenemos la opción de la plataforma BOINC, la cual a más de ser totalmente gratuita, a diferencia de PVM y MPI, brinda mejores niveles de seguridad en red, ya que protege los proyectos contra varios tipos de ataques, utilizando firmas digitales basadas en encriptación de claves públicas; también provee de mecanismos para facilitar el desarrollo de aplicaciones distribuidas. Otra de las ventajas importantes que nos ofrece BOINC, es que las aplicaciones pueden funcionar como aplicaciones BOINC con poca o ninguna modificación. Es por estas razones que el software para implementar nuestro sistema distribuido será BOINC.

CAPÍTULO 4

4 DISEÑO

En el presente capítulo se explica todo lo referente al diseño del sistema distribuido de procesamiento, en base a un conjunto de requerimientos planteados, siendo uno de los principales el uso de recursos computacionales de los equipos conectados a la red de la ESPOL, e incluso conectados a Internet. El diseño del sistema fue creado considerando que el sistema se va a basar en la plataforma distribuida BOINC.

En el transcurso de este capítulo se detalla de forma gráfica cada uno de los componentes que forman el sistema, describiendo sus características, funcionamiento, e interacción con los otros componentes del sistema.

4.1 Diseño general

4.1.1 Diseño lógico general

De manera general el sistema distribuido de procesamiento estará formado por dos partes principales, véase Fig. 4.1.1-1:

- Sistema Servidor.
- Sistema Cliente.

El sistema en general podrá administrar aplicaciones distribuidas, el **servidor** se encargará de dividir el trabajo total a realizar en unidades de trabajo pequeñas más manejables en términos de tiempo de procesamiento, distribuir estas unidades entre un grupo de computadoras **clientes**[1] o nodos dentro de una red, y controlar el procesamiento que se está realizando.

En primer lugar, para que una estación de trabajo pueda formar parte del sistema distribuido, el usuario deberá registrarse en el sistema. Este procedimiento lo podrá realizar a través de una interfaz Web o solicitándolo al administrador del sistema. Una vez que el cliente es registrado, se le asignará un identificador único que lo diferenciará de los demás usuarios y se establecerán las preferencias de

[1] Este grupo de computadoras también puede ser llamado cluster de computadoras.

procesamiento por defecto[1] para el usuario, que luego podrán ser cambiadas.

Luego de esto, el cliente deberá instalar el sistema cliente en la estación de trabajo participante, el cual solicitará al usuario la ubicación del servidor y el identificador único asignado por el servidor. Una vez que el cliente se comunica con el servidor, como lo podemos apreciar en la Figura 4.1.1-1, éste solicitará la aplicación distribuida[2] y las unidades de trabajo que debe procesar. Una vez obtenida la aplicación distribuida, el cliente comenzará a procesar las unidades de trabajo dependiendo de sus preferencias de procesamiento, mediante el uso de los ciclos ociosos del CPU preferiblemente. Cuando el cliente haya terminado el procesamiento, deberá notificar al sistema enviando el resultado obtenido al servidor para que ser analizado. Este proceso se repite cíclicamente entre todos los clientes registrados hasta que no existan unidades de trabajo que deban ser procesadas o cuando se haya llegado a una solución. El sistema deberá llevar control, de tal forma que todos sus clientes estén ocupados realizando algún procesamiento.

---

[1] Estas preferencias establecen los recursos que el usuario quiere compartir al sistema. Y el momento o situación que desea realizar el procesamiento.

[2] La aplicación distribuida que el servidor debe enviar será la específica para su plataforma.

Como podemos apreciar, el sistema total tendrá un poder computacional teórico igual a la suma del poder computacional de todos sus clientes registrados, el cual podrá ser visto como una computadora virtual capaz de procesar grandes cantidades de datos a una velocidad muy razonable.

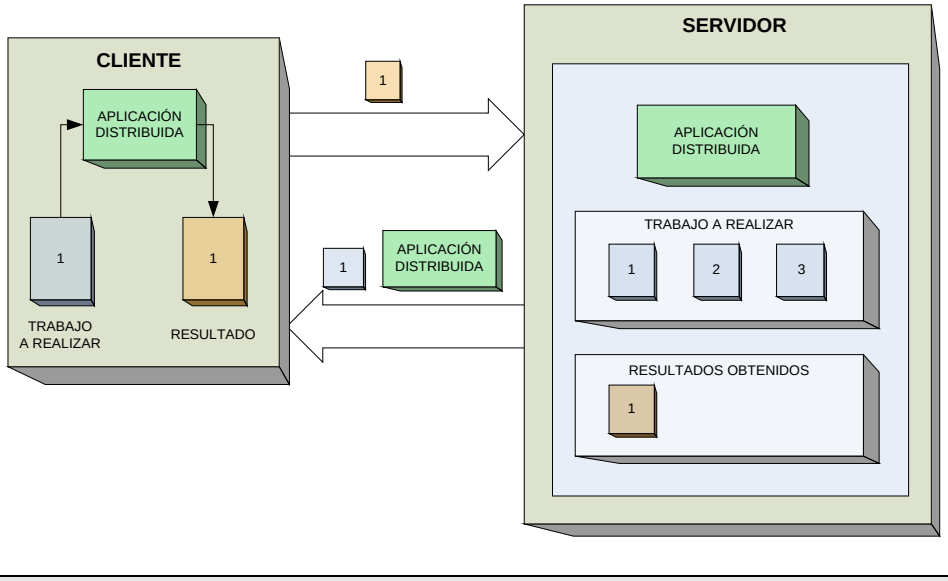


Figura 4.1.1-1. Diseño lógico general y funcionamiento del sistema.

En general, el sistema deberá contar con las siguientes características y funcionalidades:

- Proveer mecanismos de registro de nuevos usuarios.
- Ser completamente adaptable, de modo que pueda ser usado en el desarrollo de una gran cantidad de aplicaciones distribuidas.

- Ser escalable, de tal forma que al agregar una computadora más al cluster se incremente el poder computacional total del sistema.
- Dividir el trabajo a realizar, creando unidades de trabajo a ser procesadas por cada una de las computadoras disponibles que formen parte del clúster.
- Ser independiente de la plataforma, una computadora podrá formar parte del sistema como cliente sin importar su plataforma o arquitectura.
- Deberá tener un control centralizado, un servidor central será el encargado de llevar control sobre el clúster. Además llevará control de las computadoras que formen parte del clúster y de las unidades de trabajo procesadas[1].
- Ser tolerante a fallos; si una computadora que forma parte del clúster falla, éste no hará que el sistema completo colapse.
- Soportar redundancia de datos, esto es, dos o más computadoras en algún momento podrán procesar la misma unidad de trabajo. De esta manera se tratará de reducir la probabilidad de error en el procesamiento de datos.

[1] Resultados computacionales.

- Ser seguro y estar protegido de ataques que pueda sufrir. Para ello se utilizará firmas digitales basadas en encriptación de clave pública sobre todos los datos que circulen a través de la red.
- Ser configurable para que cada cliente procese trabajo en el tiempo y situación que el sistema crea conveniente, por ejemplo cuando el usuario no esté usando su computadora, aprovechando de esta forma los ciclos ociosos del CPU.
- Deberá medir el poder de procesamiento de cada cliente, mediante un test de benchmark[1], por ejemplo.

4.1.2 Diseño físico general

En cuanto al diseño físico general, como podemos apreciar en la Fig. 4.1.2-1, los clientes se comunicarán con el servidor a través de la red local del CTI, el backbone de la ESPOL, o a través del Internet.

El servidor estará conectado a la red del CTI, a través de un *switch router*; este servidor deberá tener un dominio y una IP pública para poder ser accesado desde otras redes fuera de la ESPOL. La conexión que existe de la red del CTI al backbone de la ESPOL, nos

[1] Programa especialmente diseñado para evaluar el desempeño de un sistema, de software o de hardware.

va a ayudar a que el servidor pueda ser accedido desde otras redes como el Internet.

Además de esto, el servidor deberá contar con una *tarjeta de red de alta velocidad*, lo ayudará a mejorar los tiempos de respuesta y de transferencias de archivos del servidor hacia los clientes y viceversa.

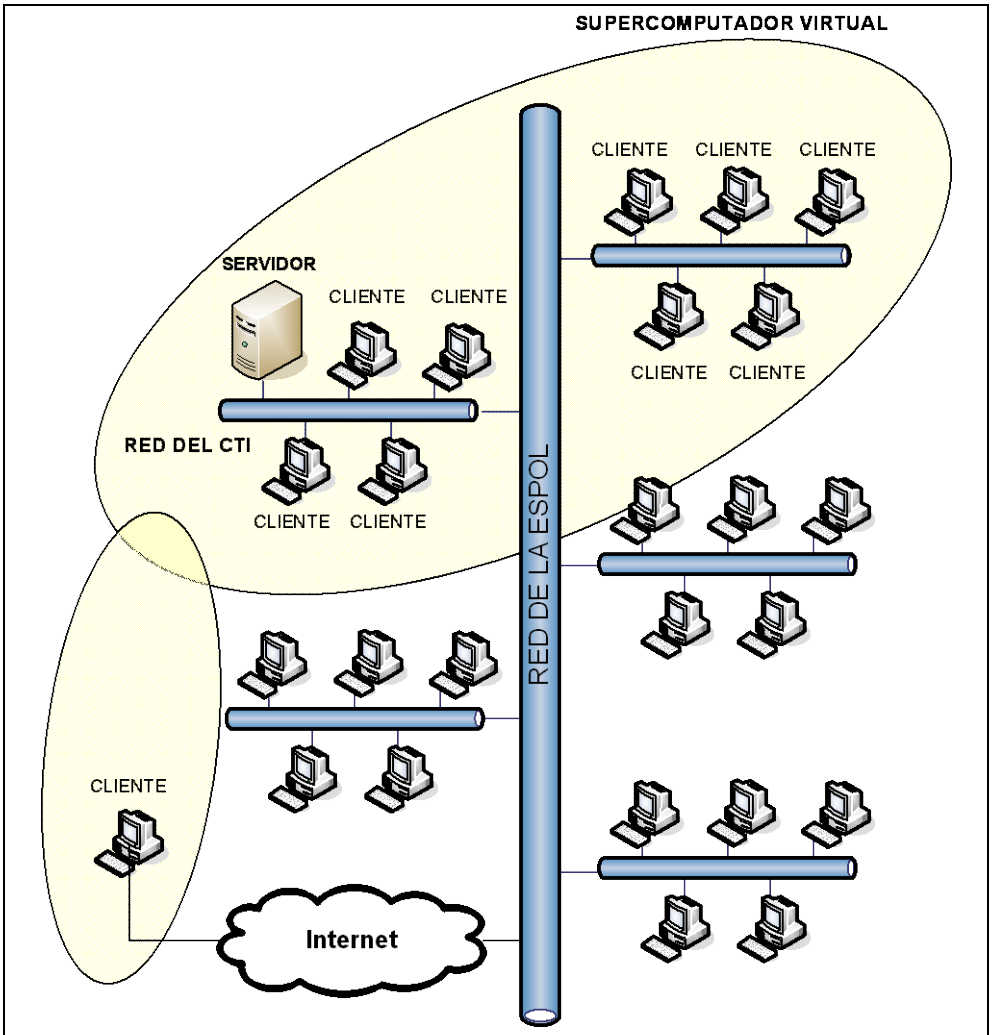


Figura 4.1.2-1. Diseño físico general del sistema.

Cada computadora cliente que va a formar parte del cluster deberá tener: una tarjeta de red que le permita conectarse al servidor; estar conectada a la red de la ESPOL o al Internet ya sea de manera directa o a través de un PROXY; y tener instalado el sistema cliente que interactuará con el servidor.

4.2 Diseño del servidor

4.2.1 Diseño lógico

Como ya mencionamos el servidor es aquel que se encargará de llevar el control sobre el cluster y controlar el procesamiento que se va a ejecutar sobre cada uno de los nodos clientes del sistema distribuido, y estará encargado de proveer las siguientes funcionalidades y características:

- Soporte para la administración de más de una aplicación distribuida, que pudieran ser desarrolladas en varias plataformas.
- Proveer mecanismos para facilitar el proceso de agregar nodos clientes al sistema.
- Crear las unidades de trabajo para ser procesadas por los clientes.
- Identificar de manera individual los clientes del sistema.
- Suministrar a los clientes la aplicación distribuida específica para su plataforma y unidades de trabajo para procesar.

- Recibir y procesar los resultados computacionales enviados por los clientes.
- Proveer un servicio de intercambio de archivos a través de un servidor de datos.
- Deberá mantener una base de datos para almacenar información de los clientes, poder computacional de cada estación, aplicaciones distribuidas, unidades de trabajos, unidades procesadas, y demás información relevante del sistema.
- Proveer un sistema de control y administración que pueda ser accedido desde el Web.
- Suministrar a los clientes un mecanismo de acceso indirecto a la información contenida en base de datos, por cuestiones de rapidez y seguridad.
- Proveer mecanismos de seguridad en la transmisión/recepción de archivos con los clientes.

Como vemos en la Fig. 4.2.1-1, el servidor estará formado por los siguientes componentes:

- Base de datos.
- Servidor de tareas.
- Servidor de datos.
- Servidor Web.

- Utilidades y programas.
- Componentes propios de una aplicación.

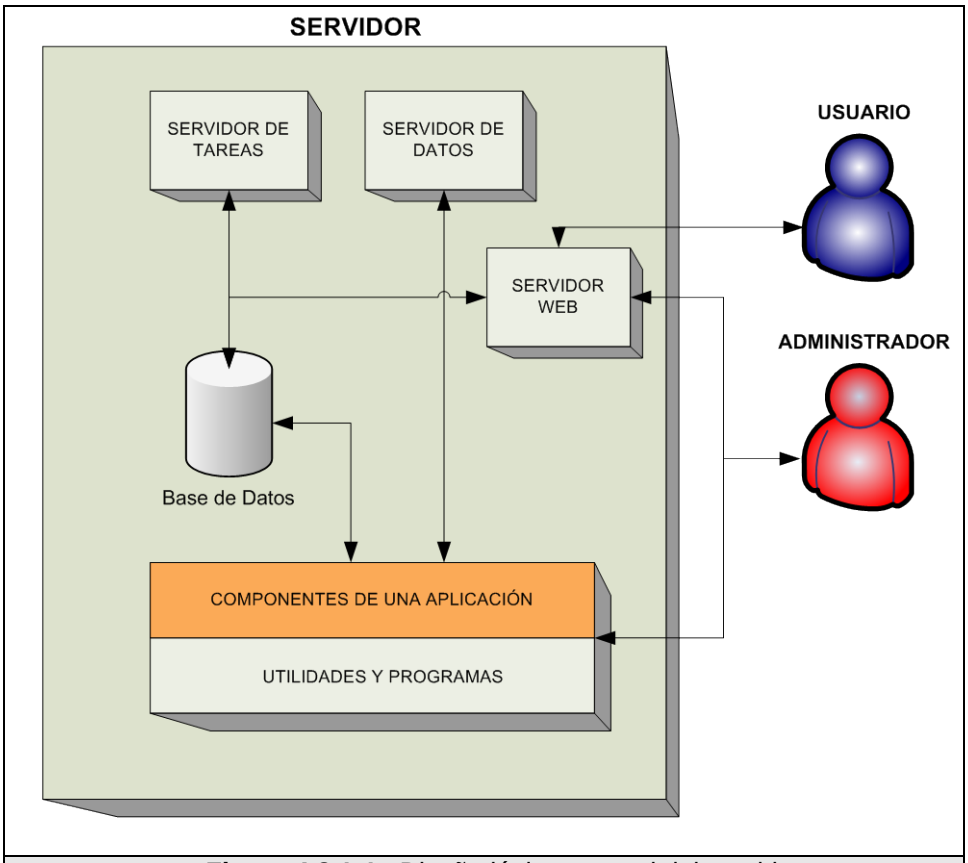


Figura 4.2.1-1. Diseño lógico general del servidor.

4.2.2 Base de datos

La base de datos almacenará información relevante del sistema tales como clientes, unidades de trabajo, resultados, aplicaciones, entre otros.

La base de datos no podrá ser manipulada directamente por los clientes, como ya se mencionó el servidor proveerá mecanismos de acceso de manera indirecta y deberá contener las siguientes tablas:

platform	Plataformas soportadas por el sistema.
app	Aplicaciones distribuidas que el sistema esta ejecutando.
app_version	Versiones de las aplicaciones. Los registros en esta tabla almacenarán la dirección donde reside el ejecutable, así como su checksum[1].
user	Usuarios del sistema.
host	Estaciones de trabajo del sistema. Los registros en esta tabla almacenarán las características más importantes de una estación de trabajo, como su poder computacional, espacio en disco, sistema operativo, memoria RAM, entre otras.
workunit	Unidades de trabajo. La descripción de la unidad de trabajo será almacenada en un archivo XML y guardada en un registro de esta tabla. Además esta tabla almacenará registros del número de resultados que se relacionan con esta unidad de trabajo, la cantidad que han sido enviados, que han sido procesados, y los que han fallado.
result	Resultados de procesamiento. Esta tabla almacenará registros de "estado" del resultado (si este ha sido procesado o no). Además tendrá otros registros que serán relevantes sólo después que un resultado haya sido procesado: tiempo de CPU usado, estado de resultado, y estado de validación.

Tabla 4.2.2-1. Tablas que va a contener la base de datos del sistema.

[1] suma de verificación o checksum es una forma de control de redundancia, una medida muy simple para proteger la integridad de datos, verificando que no hayan sido corrompidos.

4.2.3 Servidor de tareas

El servidor de tareas es aquel que va a interactuar de manera directa con el cliente distribuido. Este suministrará un conjunto de “tareas” que el cliente deberá ejecutar.

El servidor de tareas intercambiará mensajes de petición/respuesta con el servidor de tareas, a través del protocolo HTTP, usando archivos XML como mensajes. El cliente hará una petición de trabajo, y el servidor de tareas responderá con un mensaje que incluirá, entre otras cosas, lo siguiente:

- Una descripción de trabajo que debe ser procesado.
- La ubicación del servidor de datos, en donde se almacenan los archivos de entrada y salida del procesamiento. Esto es los archivos necesarios para el cómputo (la aplicación distribuida específica para su plataforma, las unidades de trabajo, librerías necesarias, entre otros); así como los resultados que se van a obtener.

Para poder enviar toda esta información, el servidor de tareas deberá realizar una serie de consultas en la base de datos. Como podemos

ver, éste será el mecanismo que se usará para no permitir al cliente un acceso directo a la base de datos.

Además de esto, el servidor de tareas deberá enviar mensajes al cliente para indicar el estado actual del sistema. Estos mensajes tendrán una "prioridad". Los mensajes con **prioridad alta** serán enviados de tal forma que el servidor de tareas se asegure que el usuario de la estación de trabajo lo lea, mediante una ventana flotante, por ejemplo. Este tipo de mensajes estarán reservados para aquellas situaciones donde este tipo de acción sea definitivamente necesaria, por ejemplo para indicar un error en la aplicación o alguna actualización de alta prioridad que el cliente deba hacer de forma manual. Por otro lado, aquellos mensajes con **prioridad baja**, podrán ser vistos por los usuarios aunque no de forma urgente; éste tipo de mensajes podrán ser almacenados en una bitácora, por ejemplo.

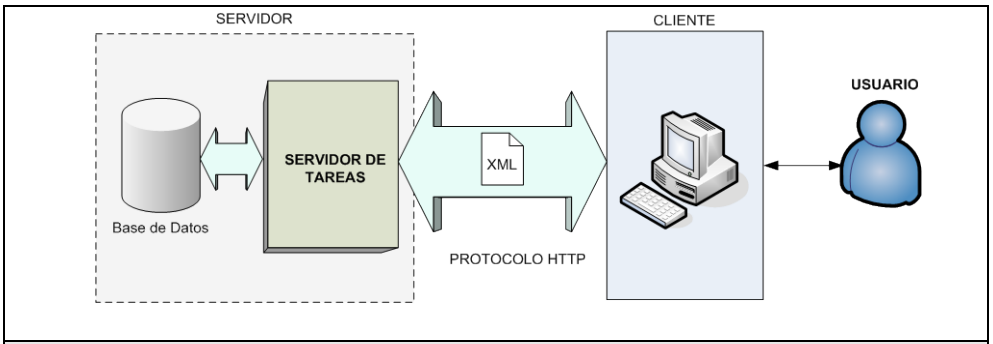


Figura 4.2.3-1. Diseño del servidor de tareas.

El servidor de tareas, además se encargará de asignar identificadores únicos a cada estación de trabajo. Cuando el cliente se conecte por primera al servidor de tareas, el servidor de tareas deberá asignarle un identificador único ID y establecer las preferencias por defecto del nuevo usuario. El servidor también deberá mantener un número de secuencia RPC para cada estación. Tanto el ID como la secuencia RPC serán almacenados en el cliente.

Si una unidad de trabajo usa más espacio en disco que un cliente tiene disponible; el servidor de tareas no deberá enviar dicha unidad de trabajo al cliente.

El servidor de tareas deberá estimar el tiempo que toma el cliente en procesar una unidad de trabajo, mediante la siguiente fórmula:

$$t = \frac{número\_de\_flops}{flops\_por\_segundo} + iops\_por\_segundo$$

Donde, *número_de_flops* representa el número de operaciones de punto flotante que se necesitan para procesar una unidad de trabajo; *flops_por_segundo* representa el número de operaciones de punto flotante que el cliente puede procesar por segundo; e

iops_por_segundo representa el número de operaciones con enteros que se necesitan para procesar una unidad de trabajo. Tanto el número de operaciones de punto flotante y de operaciones con enteros que se deben realizar para procesar una unidad de trabajo, serán calculadas en el momento el servidor cree una unidad de trabajo. Y la velocidad de cálculo o poder de procesamiento de la estación de trabajo será calculada por el sistema cliente, y enviada al servidor de tareas.

4.2.4 Servidor de datos

El servidor de datos, como vemos en la Fig. 4.2.4-1, será el encargado de almacenar; tanto los archivos necesarios para el procesamiento (aplicaciones distribuidas, unidades de trabajo, imágenes, librerías, entre otras), así como los archivos de salida o resultados que se obtengan del procesamiento.

Una vez que el servidor de tareas le indica al cliente la ubicación del servidor de datos. Éste se comunicará con el servidor de datos usando el protocolo HTTP para solicitar los archivos de entrada. El cliente solicitará los archivos necesarios para poder empezar el procesamiento, los cuales serán:

- La aplicación distribuida específica para la plataforma del cliente, que será la encargada de procesar las unidades de trabajo.
- Las unidades de trabajo.
- Los archivos propios de la aplicación, como imágenes, librerías dinámicas, librerías estáticas, y demás archivos que la aplicación distribuida requiera.

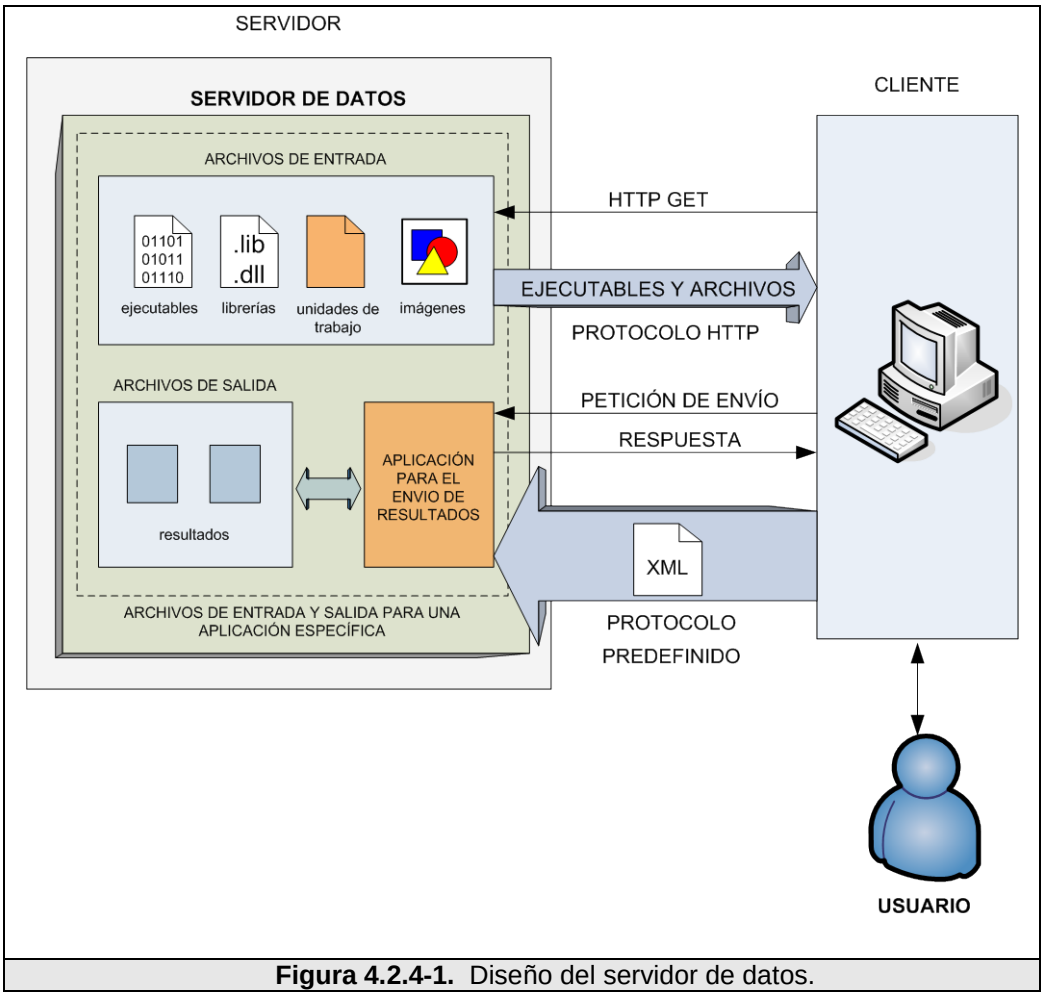


Figura 4.2.4-1. Diseño del servidor de datos.

Una vez que los archivos sean procesados el cliente deberá enviar los archivos de salida o resultados al servidor de datos. Esto se lo hará mediante el uso de una aplicación intermedia entre el cliente y el servidor, usando un protocolo predefinido a través de archivos XML; siendo éste un mecanismo de seguridad para evita el envío de archivos al servidor sin autorización. Esta acción se la realizará mediante dos operaciones:

- **Acuerdo para el envío del archivo.** El servidor recibirá un mensaje del cliente indicando que esta dispuesto a enviar el resultado obtenido de una unidad de trabajo de una aplicación en particular. El servidor debe responder si está o no dispuesto a recibir el resultado.
- **Recepción del archivo.** Una vez que el servidor esta dispuesto a recibir el archivo, éste recibirá un mensaje del cliente, que entre otras cosas, incluirá los archivos de resultados, así como también la información del archivo. Este último deberá contener una firma digital para autentificar la validez del archivo.

4.2.5 Componentes de una aplicación

Como vemos en la Fig. 4.2.5-1, los componentes de una aplicación, serán aquellos programas que ejecuten acciones de una aplicación distribuida en particular. Estas acciones serán:

- *Generar trabajo de cómputo.* Un programa se encargará de generar trabajo para una aplicación específica. Esto es, deberá dividir un trabajo grande en varios trabajos de cómputo mucho más pequeños o también llamados tareas, mucho más manejables por parte de los clientes en términos de tiempos de cómputo. La granularidad dependerá del tamaño del problema a resolver.
- *Verificar y validar los resultados de cómputo.* Una vez que un cliente envía un resultado al servidor de datos, un programa se encargará de validar dicho resultado para ratificarlo o descartarlo como válido. Esto se lo hará mediante la redundancia de datos. Si existe más de un resultado que pertenece a una misma unidad de trabajo, se hará una comparación entre todos y si la mayoría de ellos son similares se podrá decir que el resultado es válido, de lo contrario se lo tendrá que descartar.
- *Asimilar.* Este es un mecanismo por el cual se notificará al sistema la finalización del procesamiento de una unidad de trabajo, indicando si hubo un éxito o un fracaso. Este proceso será ejecutado únicamente una vez por unidad de trabajo. Si el profeso fue culminado con éxito, el sistema debe actualizar la base de datos con el nuevo resultado obtenido, por el contrario si

hubo un error durante el proceso, se deberá tomar una acción para indicar al administrador de la existencia de este error.

- *Verificar si se ha encontrado la solución al problema.* Deberá revisar los resultados para establecer si ya se encontró la solución al problema de gran desafío y notificar al administrador.

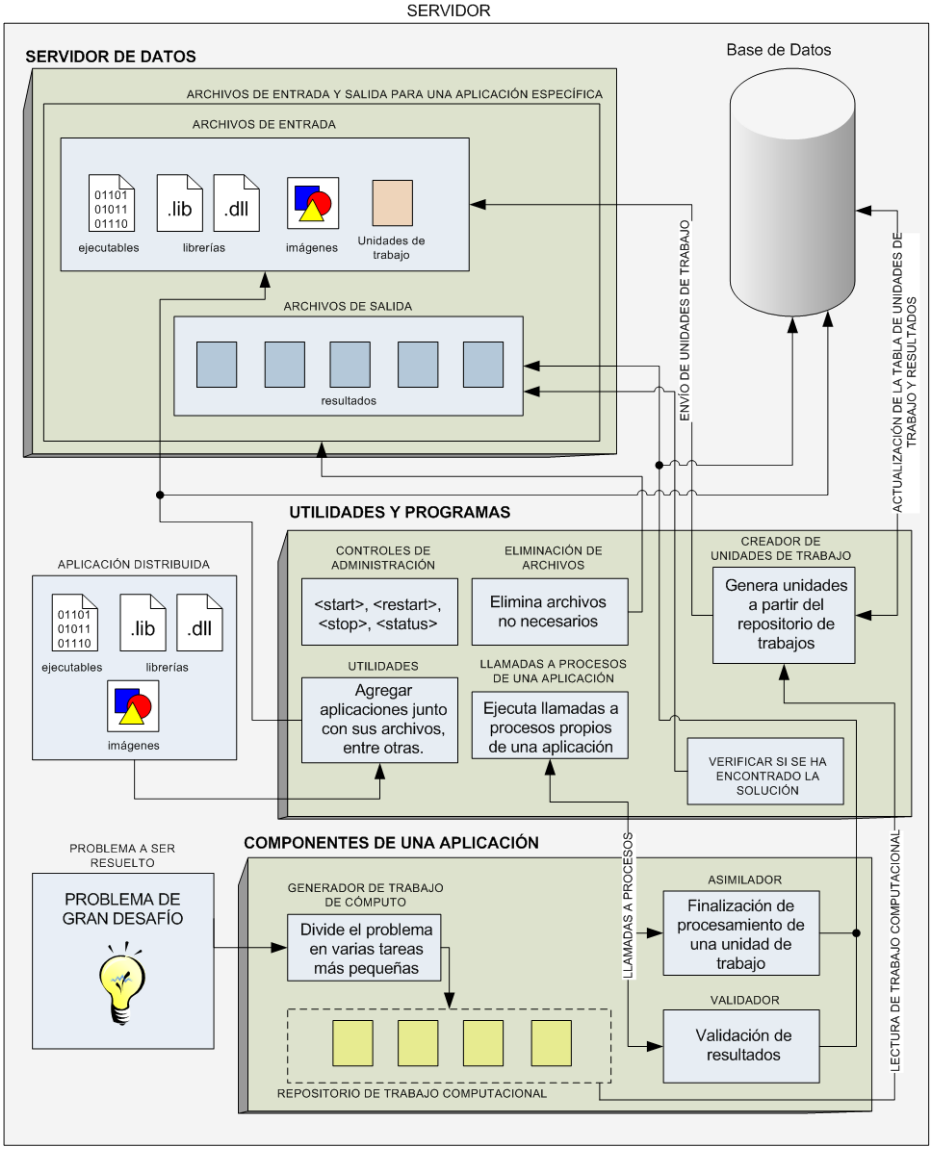


Figura 4.2.5-1. Componentes de una aplicación y utilidades y programas.

4.2.6 Utilidades y programas

Como vemos en la Fig. 4.2.5-1, las utilidades y programas serán aquellos componentes que nos ayuden a administrar el sistema en su totalidad. Deberán cumplir las siguientes funciones:

- Suministrar controles para inicializar, detener, reiniciar y verificar el estatus del servidor.
- Suministrar aplicaciones para facilitar la incorporación de aplicaciones distribuidas al sistema y archivos relacionados a ésta.
- Crear trabajo para ser procesado. Se encargará de revisar si existe trabajo que una aplicación desea procesar, si éste es el caso, generar unidades de trabajo y actualizar la base de datos con las nuevas unidades generadas. Para esto, revisa el trabajo generado por los componentes propios de la aplicación. Toma un trabajo computacional y genera una o más unidades de trabajo relacionadas a éste. Además podrá aplicar la redundancia de datos. Éste se encargará de realizar varias copias exactas de las unidades de trabajo generadas; el número de copias dependerá del nivel de redundancia que se desee conseguir. De esta forma habrá más de un cliente puede estar procesando la misma unidad de trabajo.

- Borrar los archivos que no son necesarios para prevenir errores por insuficiencia de espacio en el disco.
- Ejecutar los procesos necesarios por los componentes de una aplicación, para cada una de las aplicaciones distribuidas que el sistema está administrando.

4.2.7 Servidor Web

El servidor Web, como vemos en la Fig. 4.2.7-1, estará dividido en dos partes:

1. Interfase Web para el administrador.
2. interfase Web para los usuarios.

La interfase Web para el administrador, proveerá los siguientes servicios:

- Navegar en la base de datos, esto es, permitirá revisar los contenidos de las tablas que conforman la base de datos.
- Ver los perfiles de los usuarios.
- Ver las preferencias de procesamiento de los usuarios.
- Ver los resultados obtenidos recientemente, en las últimas 24 horas y en la última semana.
- Agregar usuarios al sistema.

La interfase Web para los usuarios, proveerá los siguientes servicios:

- Administrar su registro en el sistema, como el nombre, ciudad, correo electrónico, entre otros.
- Administrar sus perfiles.
- Administrar sus preferencias. Esta opción le permitirá cambiar al usuario, entre otras cosas, establecer el momento en que se desea que se procesen las unidades de trabajo[1], establecer el límite de espacio de disco duro usado, límites de transmisión para la carga y descarga de datos, entre otros.

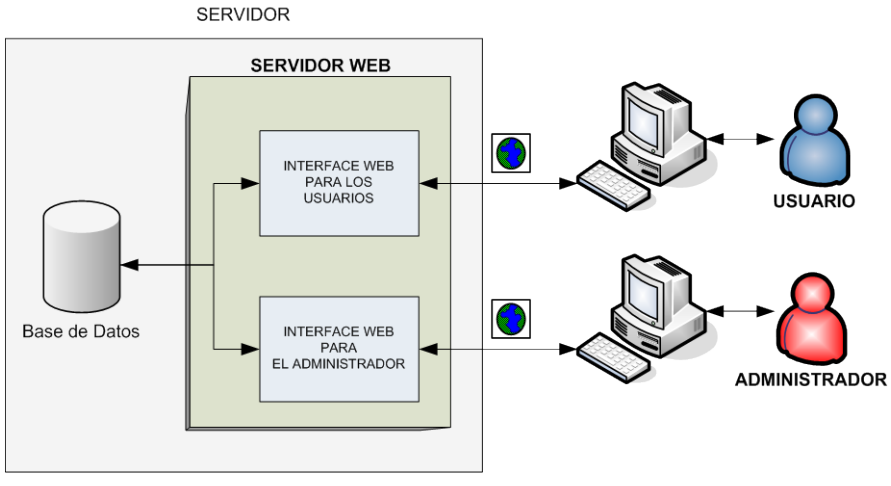


Figura 4.2.7-1. Interfase Web para el administrador y para los usuarios.

[1] El tiempo por defecto será el momento en que la computadora no se encuentre en uso.

4.3 Diseño del cliente distribuido

4.3.1 Diseño lógico

El cliente distribuido será el encargado, entre otras cosas, de procesar las unidades de trabajo, y retornar al servidor los resultados computacionales obtenidos.

Como se mencionó anteriormente, uno de los requisitos más importantes del sistema es la capacidad de soportar múltiples plataformas clientes. Para lograr esto se contará con un sistema cliente desarrollado para cada una de las plataformas o arquitecturas a las que queremos dar soporte[1]; así mismo, el servidor de datos deberá almacenar varias versiones de la misma aplicación distribuida desarrollada especialmente para cada una de éstas posibles arquitecturas. Es por esto, el sistema cliente deberá identificar y comunicar al servidor sobre la arquitectura de la estación de trabajo donde se encuentra instalado, para que el servidor le envíe sólo aquellos archivos de entrada (aplicación distribuida, imágenes, librerías, entre otros) específicos para su arquitectura.

[1] Entre las más importantes tenemos: Windows, Macintosh y Linux.

El cliente distribuido deberá cumplir con las siguientes funcionalidades y características:

- Estar desarrollado en varias plataformas a las que deseamos dar soporte.
- Comunicarse con el servidor de tareas.
- Comunicarse con el servidor de datos para solicitar los archivos de entrada y trabajo para procesar.
- Procesar trabajo usando la aplicación distribuida específica para la plataforma donde está instalado.
- Ser configurable para que procese trabajo en el momento que el cliente crea más conveniente[1], y mostrar un protector de pantalla con información relevante al procesamiento que se está ejecutando.
- Proveer una interfase gráfica para administrar el procesamiento en el cliente.
- Medir el poder computacional y recursos disponibles de la estación de trabajo cliente.
- Identificar individualmente a cada cliente; obteniendo y almacenando la información de sus características más

[1] Por omisión el momento en el que el CPU no está siendo utilizado. Esto es, usar los llamados ciclos ociosos del CPU.

importantes, como poder computacional, sistema operativo, plataforma, entre otras.

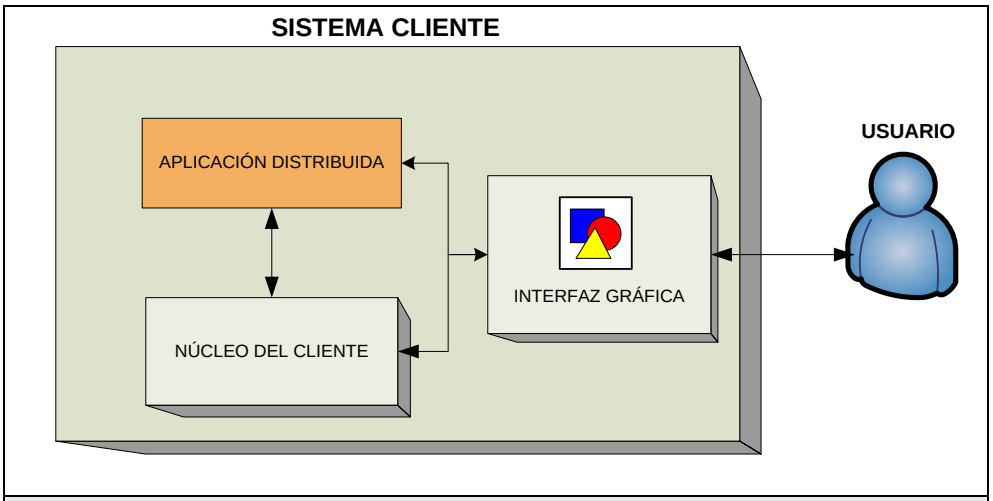


Figura 4.3.1-1. Diseño del cliente distribuido.

Como vemos en la Fig. 4.3.1-1, el sistema cliente estará formado por los siguientes componentes:

- El núcleo del cliente, que estará encargado de llevar el control sobre el procesamiento que el cliente está realizando y la comunicación entre el sistema cliente y el servidor.
- La aplicación distribuida, que estará encargada de procesar las unidades de trabajo.
- La interfaz gráfica, o administrador de trabajo del cliente, es la que estará encargada de administrar de manera gráfica el procesamiento en el cliente.

4.3.2 Núcleo del cliente

El núcleo del cliente es sin lugar a duda la parte más importante del sistema cliente. Estará encargado de la comunicación con el servidor y llevar el control sobre el procesamiento que se está ejecutando en la estación cliente.

La primera vez que el sistema cliente sea ejecutado, el núcleo del cliente deberá realizar un grupo de operaciones para obtener datos específicos de la estación de trabajo.

Este grupo de operaciones que el núcleo del cliente deberá ejecutar serán las siguientes:

1. Obtener las características generales de la estación de trabajo como: arquitectura, sistema operativo, dirección IP, nombre, espacio en disco.
2. Calcular el rendimiento del CPU. Esto se lo hará ejecutando un test de benchmark en la estación de trabajo de tal forma que se pueda obtener el número operaciones de punto flotante y operaciones con enteros que la estación cliente puede procesar por cada segundo.

3. Obtener los recursos disponibles de la estación de trabajo, como número de CPUs, espacio en disco, memoria física, memoria caché y ancho de banda disponible.

Esta información será enviada al servidor de datos, para ser almacenada y utilizada para generar y asignar un **identificador único** a la estación de trabajo cliente. Este identificador, además de ser guardado en el servidor, será enviado de regreso a la estación cliente para ser almacenado. Este paso se lo realizará sólo una vez.

Una vez identificada la estación de trabajo, el núcleo del cliente se comunicará con el servidor de tareas solicitando trabajo para ser procesado, utilizando el identificador único que le fue asignado.

El servidor de tareas deberá utilizar este identificador, para reconocer las características individuales de la estación de trabajo cliente. De esta manera, el servidor enviará sólo aquellos archivos de entrada específicos que sean adecuados para las características individuales de la estación de trabajo, como arquitectura, sistema operativo, poder de procesamiento, espacio en disco, entre otras.

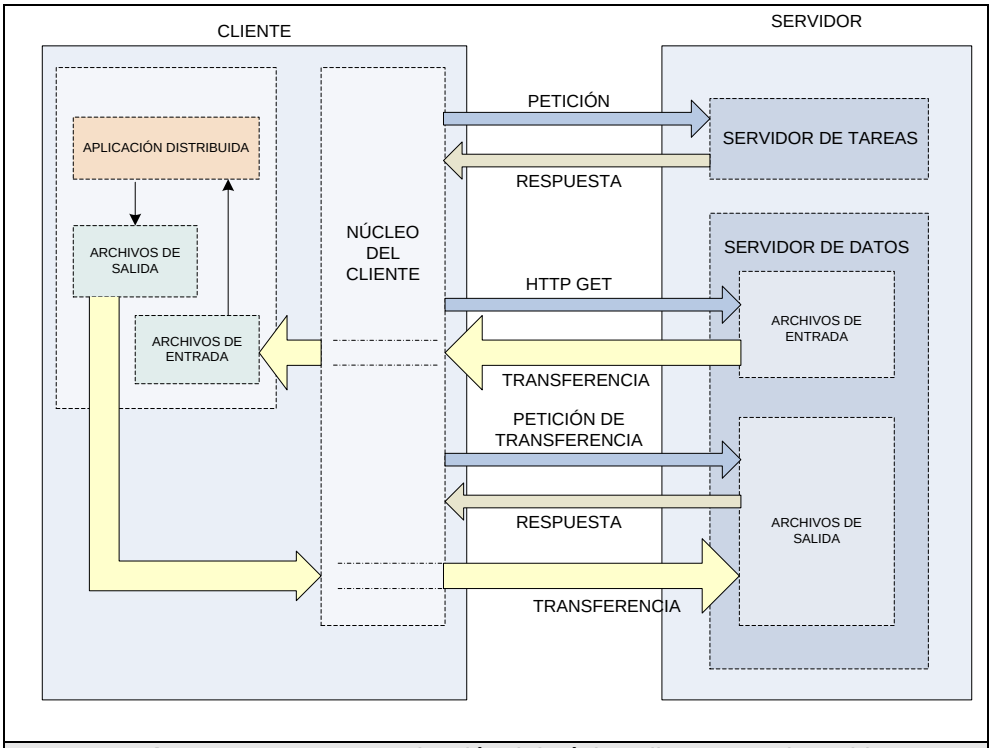


Figura 4.3.2-1. Comunicación del núcleo cliente con el servidor.

Como vemos en la Fig. 4.3.2-1, la comunicación entre el sistema cliente y el servidor se la realizará a través del núcleo cliente de la siguiente manera:

- El núcleo cliente se comunicará con el servidor de tareas a través de mensajes, solicitando trabajo para procesar.
- El servidor de tareas responderá con un mensaje que contendrá, entre otras cosas, la descripción del trabajo a ser procesado, y la ubicación del servidor de datos.
- El núcleo cliente descargará desde el servidor de datos los archivos específicos para su plataforma necesarios para iniciar el procesamiento. En primer lugar descargará la aplicación

distribuida y todos los archivos de entrada que ésta requiera, como librerías, imágenes, controles, entre otros. Luego descargará las unidades de trabajo que va a procesar haciendo uso de la aplicación distribuida. Toda la transferencia de archivos se la realizará a través del protocolo HTTP.

- Después de que la aplicación distribuida haya finalizado el procesamiento de una unidad de trabajo. El núcleo cliente deberá enviar el resultado al servidor de datos, utilizando un protocolo predeterminado con el objetivo de evitar ataques al servidor.
- Luego de esto el núcleo cliente se comunicará nuevamente con el servidor de tareas para reportar la finalización de dicha unidad de trabajo y solicitar más trabajo para procesar.

Además de que el núcleo cliente cumpla con la función de establecer una conexión con el servidor; éste deberá llevar un control sobre el procesamiento que se está llevando a cabo en el cliente. Para esto deberá cumplir con las siguientes funcionalidades:

- Comprobar por condiciones en las cuales el cliente desea suspender el procesamiento de las unidades de trabajo.
- Llevar control sobre la comunicación con el servidor.
- Iniciar la aplicación distribuida para procesar trabajo, o reiniciarla si el cómputo fue suspendido.

- Controlar si la aplicación distribuida ha sido detenida y tomar medidas al respecto.
- Inicializar la transferencia de archivos si es necesario.
- Eliminar archivos innecesarios.
- Almacenar los archivos de estado de procesamiento, si este fuera el caso.

El núcleo cliente se ejecutará en un directorio raíz. Este deberá crear y usar archivos y/o directorios que almacenen información relevante tanto de la estación de trabajo y del procesamiento que se está llevando a cabo, así como del usuario y sus preferencias de procesamiento.

Estos archivos y directorios serán los siguientes:

- Archivos de preferencias del usuario. El cual describirá las preferencias de procesamiento del usuario. Esto es, la cantidad de recursos que quiere invertir en el procesamiento de unidades de trabajo, y el momento y situación en que lo desea realizar.
- Archivo de estado de procesamiento. El cual describirá los archivos, aplicaciones, unidades de trabajo y resultados presentes en el cliente.

- Archivo del registro del usuario. El cual describirá el registro del usuario almacenado su identificador único, la ubicación del servidor de tareas, del servidor de datos y de la interfaz Web del usuario.
- Directorio de aplicaciones. El cual va a contener todos los archivos específicos referentes a una aplicación distribuida; como archivos de entrada, binarios, unidades de trabajo, resultados, entre otros.
- Directorio de procesamiento. En el cual se va a ejecutar la aplicación distribuida. Este directorio deberá contener, tanto la aplicación distribuida, así como todos los archivos de entrada que ésta necesite. Este directorio será revisado periódicamente por el núcleo cliente en búsqueda de nuevos resultados obtenidos por la aplicación.

Cuando el núcleo cliente inicie, deberá interpretar el archivo de preferencias del usuario para establecer las preferencias del procesamiento y reconocer el momento y situación que el cliente desee procesar trabajo. Si no existiera dicho archivo, el núcleo del cliente deberá comunicarse con el servidor para solicitar dichas preferencias usando el identificador único del cliente, para luego crear el archivo de preferencias del usuario.

El núcleo del cliente, para llevar el control sobre el procesamiento, deberá hacer reiteradas llamadas a la siguiente función:

```
booleano ESTADO_DEL_CLIENTE::hacer() {
    booleano accion=falso;

    si (Verificar_suspencion_de_actividad())
       retornar falso;
    accion |= Buscar_por_conexion();
    accion |= Buscar_por operacion_http ();
    accion |= Buscar_por transferencia_de_archivo();
    accion |= Comunicacion_con_servidor_de_tareas();
    accion |= Iniciar_apps();
    accion |= Manejar_apps_ejecutadas();
    accion |= Iniciar_transferencia();
    accion |= Eliminar_basura();
    Escribir_archivo_de_estado_si_es_necesario();
    retornar accion;
}
```

Esta función iniciará nuevas actividades si es necesario, o chequeará si alguna actividad ha sido completada. Esta función será llamada periódicamente desde el sistema cliente por medio de la interfaz gráfica; retornando verdadero si algún cambio ha ocurrido.

A continuación la descripción de cada una de estas funciones:

- `Verificar_suspencion_de_actividad()`, verificará aquellas condiciones; ingreso por teclado o ratón, baterías bajas, o cualquiera en las cuales las preferencias del usuario establecen que no se debe realizar trabajo alguno.

- `Buscar_por_conexión(),Buscar_por_operacion_http(),Buscar_por_transferencia_de_archivo(),` consultará posibles conexiones que el cliente desea realizar con el servidor.
- `Iniciar_app(),` verificará si es posible dar inicio a una aplicación; por ejemplo, si el CPU está vacante y existe trabajo que puede ser procesado, se dará inicio a la aplicación.
- `Manejar_app_ejecutadas(),` verificará si una aplicación que se está ejecutando ha finalizado, y si es así eliminará todo registro de ella.
- `Iniciar_transferencia(),` dará inicio a una transferencia si es necesario.
- `Eliminar_basura(),` verificará si hay objetos que pueden ser descartados o eliminados.
- `Escribir_archivo_de_estado_si_es_necesario(),` cualquiera de las funciones mencionadas anteriormente podrían cambiar el estado de tal forma que se deba modificar el archivo estado del cliente. Esta función realizará cambios en este archivo, si así lo crea necesario.

4.3.3 Aplicación distribuida

La aplicación distribuida, será la encargada de procesar las unidades de trabajo provenientes del servidor de tareas. El núcleo del cliente deberá comunicarse con el servidor de tareas para solicitar la aplicación distribuida más adecuada para las características de la estación cliente.

Como vemos en la Fig. 4.3.3-1, el núcleo del cliente almacenará la aplicación distribuida junto con los archivos de entrada en el directorio de aplicaciones. El núcleo del cliente deberá leer las preferencias del usuario a establecer el momento o situación para empezar. Cuando el cliente desee comenzar con el procesamiento de unidades de trabajo, el núcleo del sistema deberá hacer una copia de la aplicación distribuida junto con sus archivos de entrada en el directorio de procesamiento, para luego ejecutarla.

Una vez ejecutada la aplicación distribuida, leerá tanto las unidades de trabajo, así como los demás archivos que necesite para su ejecución, como librerías, imágenes, controles, entre otros. La aplicación distribuida deberá tener una comunicación permanente, tanto con la interfaz gráfica, así como con el núcleo del cliente para informar en cada momento del estado actual del procesamiento.

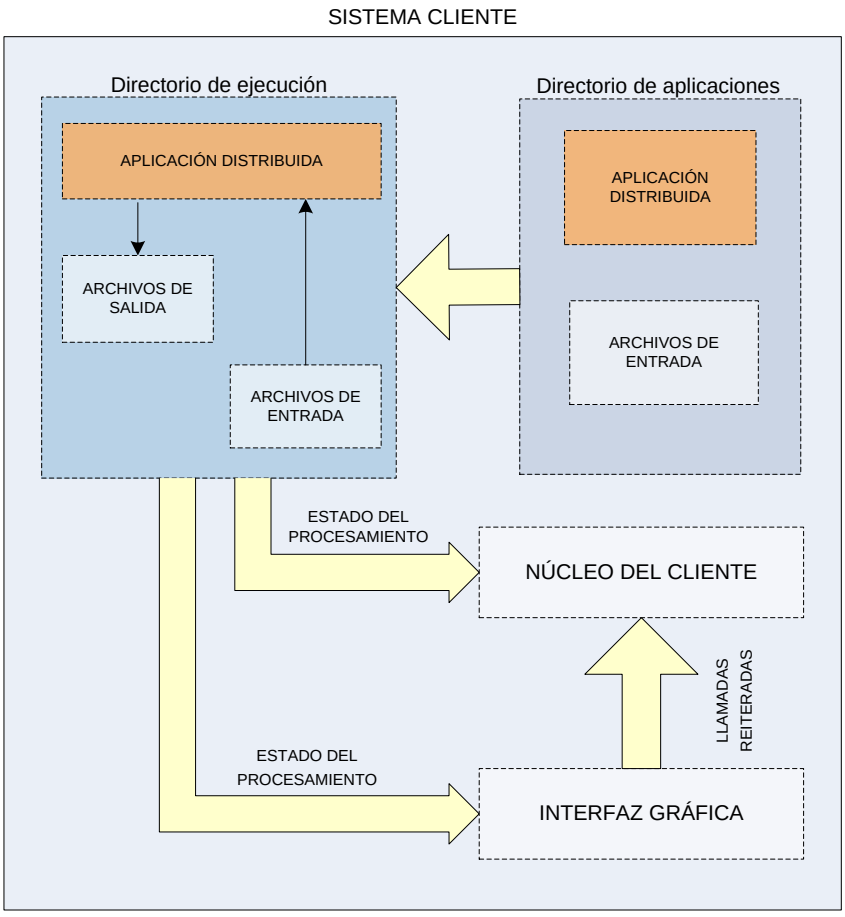


Figura 4.3.3-1. Diseño de la aplicación distribuida.

Como vemos en la Fig. 4.3.3-2, la aplicación distribuida como tal, estará dividida en dos componentes:

- Módulo de procesamiento. El cual se encargará de procesar las unidades de trabajo y retornar un resultado computacional.
- Módulo de presentación gráfica. El cual se encargará de mostrar información de manera gráfica respecto al procesamiento que el cliente está ejecutando. Además tendrá la capacidad de usar el protector de pantalla para mostrar ésta información gráfica.

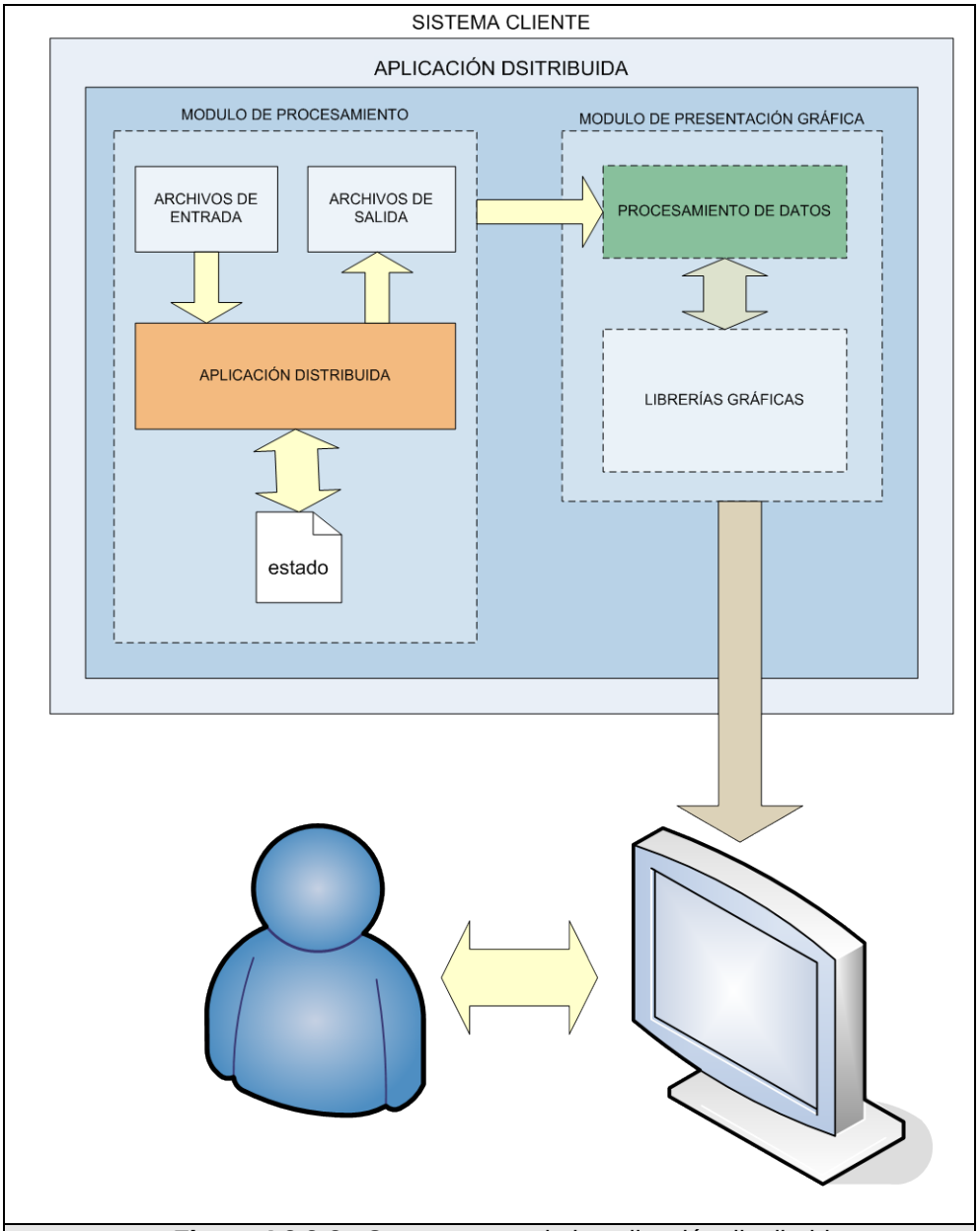


Figura 4.3.3-2. Componentes de la aplicación distribuida.

Cuando el cómputo sea suspendido o interrumpido, la aplicación distribuida deberá crear en un archivo de estado, de tal forma que éste almacene el progreso del procesamiento de la unidad de trabajo que ha ejecutado hasta ese momento. De esta manera, cuando la

aplicación distribuida sea ejecutada nuevamente por el núcleo cliente; antes de iniciar con el procesamiento de una unidad de trabajo, deberá consultar si existe un estado previo de procesamiento, el cual indique que debe continuar con un procesamiento que fue interrumpido o suspendido.

El módulo de presentación gráfica de la aplicación distribuida, deberá generar gráficos representativos del procesamiento que el cliente está realizando, como progreso de cómputo de la unidad de trabajo, estado de procesamiento, y demás información que se considere importante o interesante para mostrar de forma gráfica

Estos gráficos serán mostrados a manera de protector de pantalla en la estación cliente, esto es, sólo podrán ser visualizados en el momento que la aplicación distribuida este procesando unidades de trabajo haciendo uso de los ciclos ociosos del CPU.

Tanto el módulo de procesamiento de unidades de trabajo y el de presentación gráfica, estarán ejecutándose al mismo tiempo de forma paralela; a diferencia que el segundo se hará efectivo siempre y cuando el protector de pantalla esté siendo presentado.

4.3.4 Interfaz gráfica

La interfaz gráfica también cumplirá un papel importante dentro del sistema cliente. Estará encargada de hacer reiteradas llamadas al núcleo cliente para comenzar su ejecución, y a su vez llevar control sobre las acciones que éste se encuentre realizando en la estación de trabajo.

La interfaz gráfica deberá cumplir con las siguientes funcionalidades:

- Ejecutar el núcleo del cliente.
- Mostrar la comunicación que el núcleo del cliente esté llevando a cabo con el servidor de tareas.
- Mostrar la transferencia de archivos que el núcleo cliente esté llevando a cabo con el servidor de datos.
- Mostrar los recursos disponibles de la estación de trabajo cliente.
- Mostrar el estado de la aplicación distribuida.
- Mostrar el progreso de procesamiento de las unidades de trabajo, el tiempo utilizado y el tiempo estimado para la finalización.

CAPÍTULO 5

5 IMPLEMENTACIÓN

En el presente capítulo se hace una descripción detallada de los pasos seguidos para la instalación y configuración del sistema de procesamiento distribuido, en base al diseño propuesto en el capítulo anterior. Se describe la instalación y configuración de la plataforma distribuida y sus clientes distribuidos. Además se describe el análisis, diseño e implementación de una aplicación distribuida que es probada sobre el sistema.

5.1 Implementación del servidor

Para la implementación del servidor se buscó una computadora genérica que cuente con las siguientes características:

- Procesador Pentium IV de 2.8Ghz o superior.
- Tarjeta madre Intel.
- Lector de CD.
- Disquetera.
- Tarjeta de red de alta velocidad Gigabit Ethernet.
- Disco de 80GB SATA o superior.
- Memoria RAM de 1GB o superior.

Todas estas características se basan en las necesidades que el sistema servidor requiere y en las funcionalidades que éste debe proveer al sistema completo.

Por ejemplo, notamos que toda comunicación con los clientes se la realiza a través de la red, lo que significa que el servidor deberá contar con una tarjeta de red de alta velocidad de transmisión que asegure de una u otra forma la calidad del servicio.

5.1.1 Características del servidor

Para la instalación del sistema servidor se utilizó una computadora genérica (véase Fig. 5.1.1-1), con las siguientes características:

- Tarjeta madre marca Intel modelo 875PBZ, bus de 800Mhz.
- Procesador Pentium IV 3.2 Ghz HT.
- Disco duro marca Samsung modelo Barracuda 7200.7 SATA de 120 GB.
- Tarjeta de Red Gigabit Ethernet 10/100/1000 Mbps.
- Memoria RAM de 1 GB, repartida en 4 módulos de memoria DDR400 de 256MB, para activar el canal dual.

Figura 5.1.1-1 (a) Computadora utilizada como servidor. (b) Vista interna del servidor.

Para más información acerca de las características del servidor, así como precios de las partes consulte el Apéndice D.

5.1.2 Instalación del servidor

El sistema operativo que se instaló en el servidor fue Fedora Core 2, descargado desde el sitio Web oficial del Proyecto Fedora (http://download.fedora.redhat.com/pub/fedora/linux/core/2/i386/iso/).

Los pasos más importantes que se deben considerar en el proceso de instalación son:

1. Utilizar un tipo de instalación de Personalizado, de esta forma se pueden seleccionar los paquetes para instalar. véase Fig. 5.1.2-1**.**

Figura 5.1.2-1. Pantalla de selección del Tipo de instalación.

2. Seleccionar sólo los siguientes grupos de paquetes para la instalación:
   - **Sistema X Window.** Grupo de paquetes para utilizar la interfaz gráfica del sistema operativo.

- **Entorno de escritorio GNOME.** Interfaz gráfica de escritorio GNOME.
- **Herramientas de configuración del servidor.** Herramientas de configuración propias del sistema operativo.
- **Servidor Web.** Seleccionar los siguientes paquetes adicionales:
- **Servidor de Correo.** Paquetes para el servidor de correos.
- **Servidor FTP.** Paquetes para el servidor ftp y sftp.
- **Herramientas de desarrollo**. Principales herramientas de desarrollo de aplicaciones.
- **Desarrollo de software para X**. Paquetes para el desarrollo de aplicaciones en X Window.
- **Desarrollo de software anticuado**. Paquetes para soportar compatibilidad con sistemas anteriores.
- **Herramientas del sistema.** Seleccionar los siguientes paquetes adicionales:
- **Herramientas de administración**. Colección de herramientas gráficas de administración.

3. La red debe ser configurada de tal forma que el servidor posea nombre de dominio, una IP pública, así como una puerta de enlace y servidores de dominio (DNS) que le permitan tener acceso al Internet, véase Fig. 5.1.2-2.

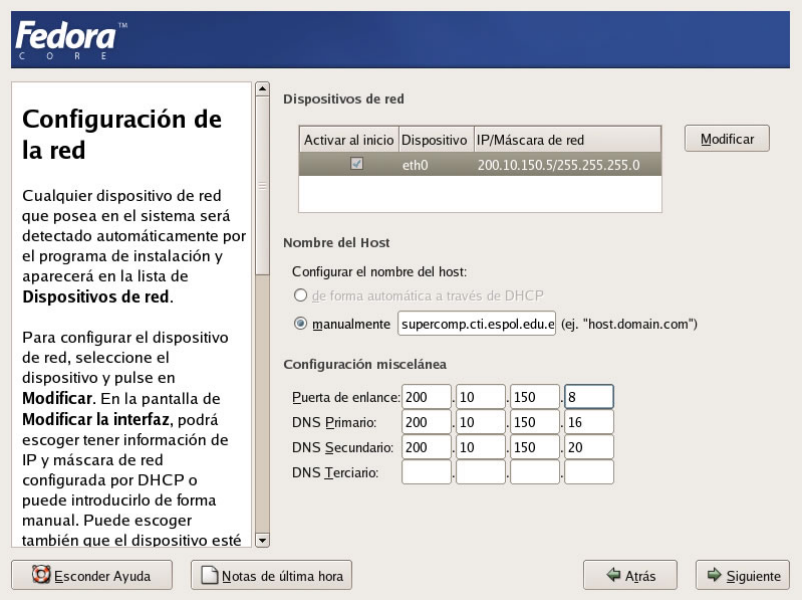


Figura 5.1.2-2. Pantalla de configuración de la red.

En la tabla 5.1.2-1, podemos ver los datos de configuración del la red que fueron colocados en el servidor.

Nombre del host	supercomp.cti.espol.edu.ec
IP	200.10.150.5
Máscara de red	255.255.255.0
Puerta de enlace	200.10.150.8
DNS Primario	200.10.150.16
DNS Secundario	200.10.150.20

Tabla 5.1.2-1. Valores de configuración para la red en el servidor.

Para más información acerca de la instalación del sistema operativo consulte el Anexo B.1.

A continuación se instaló la base de datos MySQL, junto con el cliente, librerías compartidas de MySQL y las librerías para el desarrollo de aplicaciones MySQL. La base de datos instalada fue la versión 4.0.20. Para instalar la base de datos se debe ejecutar la siguiente línea de comando:

```
$rpm –iU MySQL-server-4.0.20-0.i386.rpm
```

El cliente MySQL instalado fue la versión 4.0.20. Para instalar el cliente MySQL se debe ejecutar la siguiente línea de comando:

```
$rpm –iU MySQL-client-4.0.20-0.i386.rpm
```

Una vez instalados tanto el servidor, como el cliente MySQL se instalaron las librerías compartidas de MySQL: compat y shared-compat, ambas en la versión 4.0.20. Para instalar éstas librerías se deben ejecutar las siguientes líneas de comando:

```
$rpm –iU MySQL-shared-4.0.20-0.i386.rpm
$rpm –iU MySQL-shared-compat-4.0.20-0.i386.rpm
```

A continuación, se instalaron las librerías para desarrollo de aplicaciones MySQL versión 4.0.20. Para instalar éstas librerías se deben ejecutar las siguientes líneas de comando:

```
$rpm –iU MySQL-devel-4.0.20-0.i386.rpm
```

Luego de haber instalado todos los paquetes referentes a la base de datos MySQL, se instaló lo siguiente: el módulo de MySQL para PHP; y el módulo de MySQL para python.

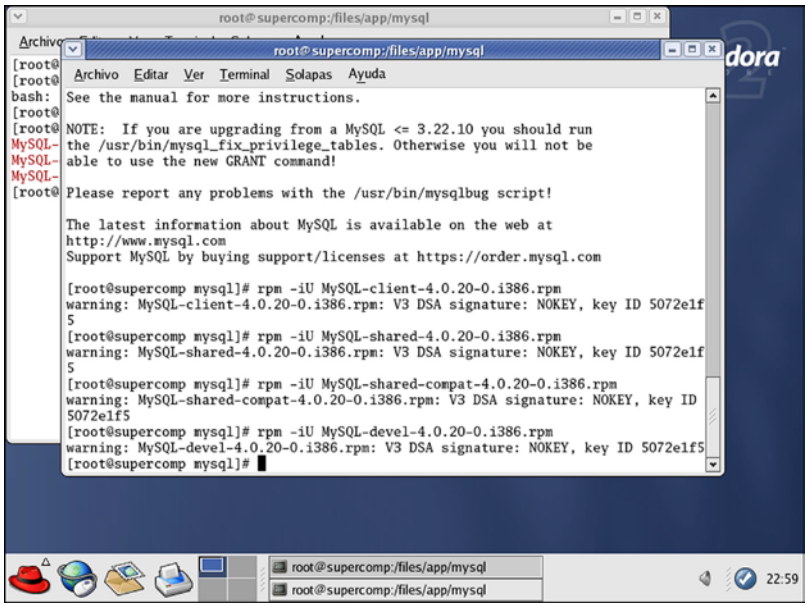


Figura 5.1.2-3. Instalación de MySQL server, cliente, librerías compartidas y librerías de programación.

El módulo de MySQL para PHP instalado fue la versión 4.3.4 distribución 11. Para instalar éste módulo se debe ejecutar la siguiente línea de comando:

```
$rpm –iU php-mysql-4.3.4-11.i386.rpm
```

La versión instalada del modulo MySQL para python fue la 1.0.0. Para instalar éste módulo, primero se debe descomprimir el archivo usando la siguiente línea de comando:

```
$tar –xzvf MySQL-python-1.0.0.tar.gz
```

Este comando creará una carpeta llamada MySQL-python-1.0.0 en el mismo directorio. El paso siguiente, es entrar en dicha carpeta para luego ejecutar las siguientes líneas de comando:

```
$python setup.py build
$python setup.py install
```

En cuanto a la instalación física, el servidor fue colocado en la cabina de servidores del departamento técnico del CTI y conectado al switch 3Com junto con los demás servidores. Éste switch a su vez, está conectado al switch principal del CTI ubicado en el Aula Satelital, véase la Fig. 5.1.2-10.

Figura 5.1.2-4. (a) Computadora servidor en la cabina, junto con el switch 3Com. (b) Rack ubicado en el Aula Satelital donde se encuentra el switch principal del CTI.

La red del CTI se conecta al Internet a través del backbone de la ESPOL. De esta forma, el servidor tiene acceso al Internet. En la Fig. 5.1.2-11, podemos ver el diagrama de la conexión del servidor con la red del CTI, y ésta a su vez al Internet.

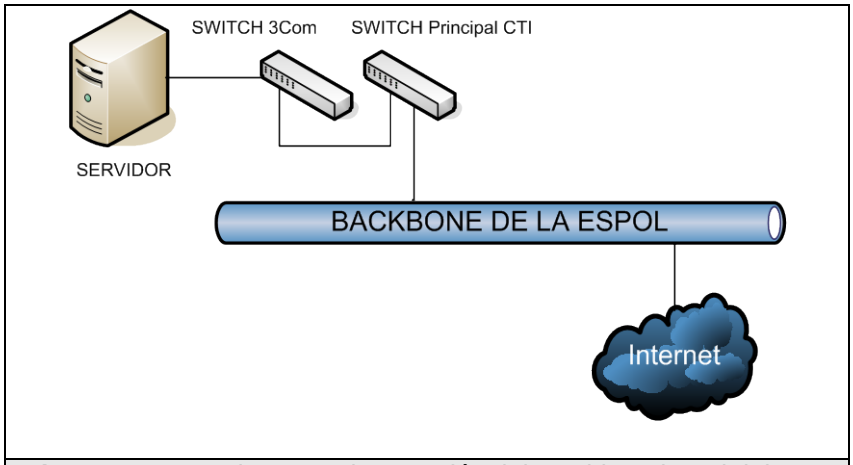


Figura 5.1.2-5. Diagrama de conexión del servidor a la red del CTI.

5.1.3 Configuración del servidor

Una vez que le sistema operativo junto con parte de los paquetes necesarios fueron instalados; y el servidor conectado a la red del CTI, se dio inicio a la configuración del servidor.

El primer paso en la configuración del servidor, fue la configuración del sistema operativo para que inicie en modo consola. Para hacer esto, se debe editar el archivo */etc/inittab* en la siguiente línea:

```
id:5:initdefault:
```

Y cambiar el valor 5 por el nuevo valor de 3, para poder especificar que el inicio por defecto del sistema operativo sea en modo consola. Al final, la línea debe quedar de la siguiente manera:

```
id:3:initdefault:
```

De esta forma la próxima vez que el sistema operativo sea iniciado, se mostrará en modo consola.

El siguiente paso fue la configuración del firewall, de tal forma que sólo permita conexiones con el Internet a través de los puertos: 22 para ssh, 80 para htttp y 443 para https; así como también permita conexiones locales con los puertos 3306 para MySQL y 21 para sendmail.

PUERTO	SERVICIO	PERMISOS	DESCRIPCION
22/tcp	ssh	Abierto para Internet	Puerto para la conexión remota con el servidor.
80/tcp	http	Abierto para Internet	Puerto para conexiones http. Apache usa este puerto.
443/tcp	https	Abierto para Internet	Puerto para conexiones https. Apache con conexión segura ssl usa este puerto.
3306/tcp	mysql	Abierto localmente	Puerto para conexiones a la base de datos MySQL.
25/tcp	smtp	Abierto localmente para enviar y totalmente para recibir	Puerto para envió de correo. Sendmail usa este puerto.

Tabla 5.1.3-1. Detalle de los puertos abiertos en el servidor.

En la Tabla 5.1.3-1 podemos apreciar en detalle la descripción de estos puertos.

Para aplicar estas reglas en el firewall se debe crear un archivo en la carpeta */etc* llamado *firewall.sh*; el cual contendrá los comandos que se deben aplicar en el firewall.

Este archivo debe contener las siguientes líneas de código:

- Declaración de variables que usará el script:

```
if_principal=eth0
red_cti=200.10.150.0/24
ip1=200.10.150.5/32
internet=0/0
lo=127.0.0.1/32
```

- Eliminar cualquier regla existente en el firewall:

```
iptables -F
iptables -F INPUT
iptables -F OUTPUT
iptables -F FORWARD
iptables -F -t mangle
iptables -F -t nat
iptables -X
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
```

- Reglas locales:

```
# Que se pueda conectar a si mismo
iptables -A INPUT -s $lo -d $lo -j ACCEPT
iptables -A OUTPUT -s $lo -d $lo -j ACCEPT

iptables -A INPUT -s $ip1 -d $ip1 -j ACCEPT
iptables -A OUTPUT –s $ip1 -d $ip1 -j ACCEPT
```

- Reglas para el servidor Internet:

```
# Resolucion de nombres (DNS)
iptables -A OUTPUT -s $ip1 -d $internet -p udp --dport 53 -j
ACCEPT

# Permita conexiones HTTP para IP1
iptables -A INPUT -s $internet -d $ip1 -p tcp -m multiport --dport
80,443 -j ACCEPT

iptables -A OUTPUT -s $ip1 -d $internet -p tcp -m multiport --
sport 80,443 -j ACCEPT


# Que pueda navegar (HTTP, HTTPS)
iptables -A OUTPUT -s $ip1 -d $internet -p tcp -m multiport --
dport 80,443 -j ACCEPT

iptables -A INPUT -s $internet -d $ip1 -p tcp -m multiport --sport
80,443 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

- Reglas para ssh:

```
# Permita conexiones ssh
iptables -A INPUT -s $internet -d $ip1 -p tcp --dport 22 -j ACCEPT

iptables -A OUTPUT -s $ip1 -d $internet -p tcp --sport 22 -j
ACCEPT


# Que pueda hacer ssh
iptables -A OUTPUT -s $ip1 -d $internet -p tcp --dport 22 -j
ACCEPT

iptables -A INPUT -s $internet -d $ip1 -p tcp --sport 22 -m state
--state ESTABLISHED,RELATED -j ACCEPT
```

- Reglas para el envío de correo:

```
# Enviar correo SMTP
iptables -A OUTPUT -s $ip1 -d $internet -p tcp --dport 25 -j
ACCEPT

iptables -A INPUT -s $internet -d $ip1 -p tcp --sport 25 -j
ACCEPT
```

- Reglas para el ping:

```
# Que pueda hacer ping pong
iptables -A OUTPUT -s $ip1 -d $internet -p icmp --icmp-type echo-
request -j ACCEPT
iptables -A INPUT -s $internet -d $ip1 -p icmp --icmp-type echo-
reply -j ACCEPT

# Que le puedan hacer ping pong para IP1
iptables -A INPUT -s $internet -d $ip1 -p icmp --icmp-type echo-
request -j ACCEPT
iptables -A OUTPUT -s $ip1 -d $internet -p icmp --icmp-type echo-
reply -j ACCEPT
```

- Reglas para el MySQL:

```
# MySQL (solo para red CTI)
iptables -A INPUT -s $red_cti -d $ip1 -p tcp --dport 3306 -j
ACCEPT
iptables -A OUTPUT -s $ip1 -d $red_cti -p tcp --sport 3306 -j
ACCEPT
```

El archivo completo de configuración del firewall, se encuentra en el Apéndice A.

A continuación se deben otorgar permisos de ejecución al archivo *firewall.sh* usando la siguiente línea de comando:

```
$chmod 700 firewall.sh
```

Se debe de asegurar que el servicio firewall este levantado. Para iniciar el firewall se debe ejecutar la siguiente línea de comando:

```
$service iptables start
```

Posteriormente se ejecuta el archivo, usando la siguiente línea de comando:

```
$./firewall.sh
```

Una vez ejecutada ésta última línea de código, que todas las reglas existentes en el firewall son eliminadas y las nuevas reglas especificadas en el archivo son aplicadas. A continuación se debe grabar éstas nuevas reglas utilizando la siguiente línea de comando:

```
$service iptables save
```

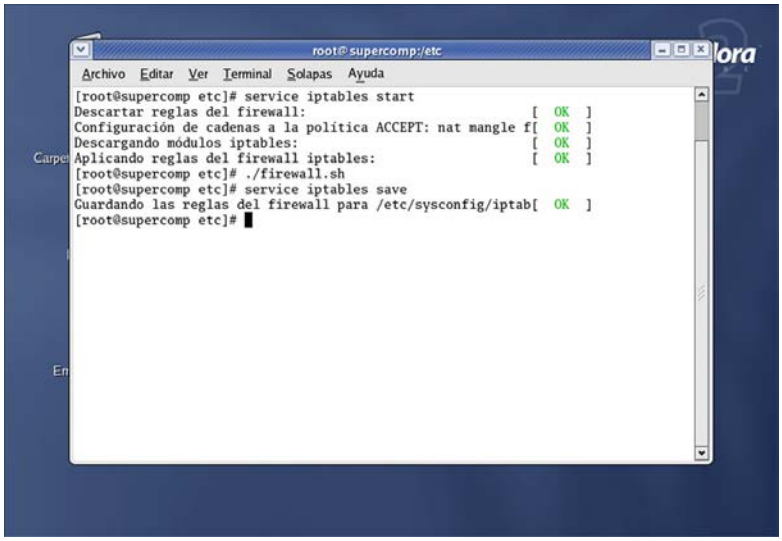


Figura 5.1.3-1. Configuración del firewall.

En la Fig. 5.1.3-1, se puede apreciar la secuencia de comandos que fueron ejecutados en la consola.

El siguiente paso en la configuración del servidor, fue la configuración de los servicios de Apache y de MySQL; cada uno de estos identificados como **httpd** y **mysql**, respectivamente; para que sean iniciados junto con el sistema operativo.

Para configurar el sistema operativo de tal forma que inicie éstos servicios al encender el equipo, se deben ejecutar las siguientes líneas de comando:

```
$chkconfig mysql on
$chkconfig httpd on
```

A continuación se deben levantar ambos servicios, ejecutando las siguientes líneas de comando:

```
$service httpd start
$service mysql start
```

Por último se otorgaron permisos al usuario root para que pueda modificar las tablas de la base de datos MySQL desde la IP que le fue asignada al servidor.

Para lograr esto, primero debemos ingresar a la consola de MySQL ejecutando la siguiente línea de comando:

```
$mysql –u root
```

Una vez dentro de la consola debemos ejecutar la siguiente línea de código:

```
$mysql> grant all on *.* to root@200.10.150.5;
```

Para salir de la consola de MySQL ejecutamos la siguiente línea de comando:

```
$mysql> quit;
```

5.1.4 Instalación de BOINC

La versión de la plataforma BOINC que se utilizó la versión 3.04, lanzada en Mayo del 2004, la cual fue instalada en el servidor.

Para instalar BOINC en el servidor, primero se debe descomprimir el archivo en la raíz (/) del sistema operativo, usando el siguiente comando:

```
$tar –xzvf boinc-2004-cvs-2004-05-09.tar.gz
```

Una vez ejecutado este comando se creará una carpeta llamada `boinc` en el directorio raíz. El paso siguiente es entrar en este directorio y ejecutar el siguiente comando:

```
$./configure
```

Este comando configurará y creará los archivos de BOINC para la compilación de la plataforma. El siguiente paso es la compilación y la verificación de que la plataforma está funcionando de manera correcta. Estas dos acciones se las puede realizar al mismo tiempo ejecutando el siguiente comando dentro del directorio `boinc`:

```
$make check
```

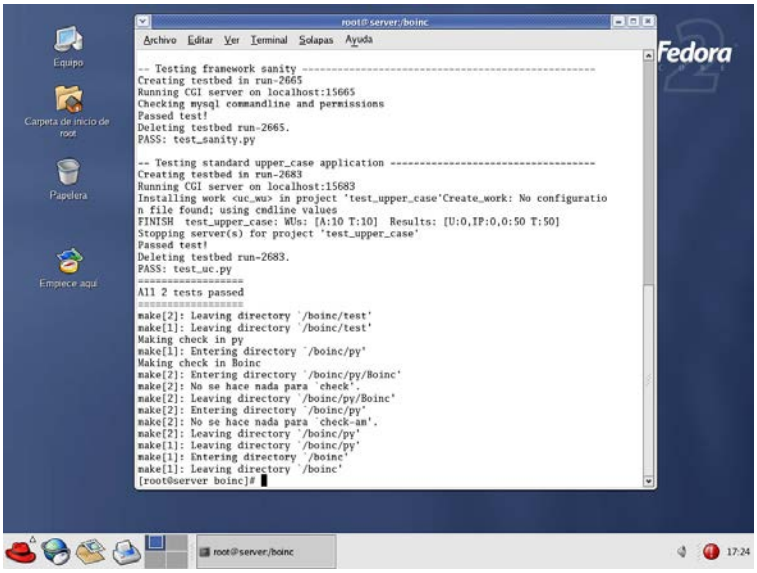


Figura 5.1.4-1. Instalación de BOINC.

En la Fig. 5.1.4-1, se puede apreciar los resultados que se deben obtener cuando se ejecuta éste último comando.

5.1.5 Configuración de BOINC

Una vez que la plataforma BOINC fue instalada en el servidor. Lo siguiente que se hizo fue la configuración. Cabe señalar que la plataforma BOINC como tal no es de mucha utilidad sin la existencia de proyectos distribuidos, tema que será tratado posteriormente.

Cuando se instala la plataforma BOINC, sólo se compilan los archivos fuentes y se verifica si el servidor está configurado para que los proyectos que serán creados posteriormente puedan trabajar correctamente; emulando situaciones en que varios clientes soliciten, luego procesen y por último envíen los resultados al servidor.

Existen archivos de prueba en la carpeta `test` del directorio de BOINC, de tal forma que si se desea realizar alguna de estas pruebas, sólo se deba ejecutar el archivo de la prueba respectiva, por ejemplo:

```
$./test_uc.py
```

Por otro lado, cuando se crea un proyecto en BOINC, los archivos de la carpeta `html` son copiados al directorio del nuevo proyecto ubicado en la carpeta `projects` del directorio de BOINC.

El objetivo de la configuración de la plataforma es modificar los archivos de la carpeta `html` ajustándolos a las necesidades propias del sistema, logrando de ésta manera que cuando se creen nuevos proyectos en la plataforma, ya cuenten con estos archivos modificados.

Lo primero que se debe hacer es editar el archivo `util.inc`, ubicado en el directorio `boinc/html/inc`, reemplazando la siguiente función:

```
function send_auth_email($email_addr, $auth) {
   mail($email_addr, PROJECT." new account confirmation",
   "This email confirms the creation of your ".PROJECT."
   account.

   ".PROJECT." URL:          ".MASTER_URL."
   Your account ID:             $auth\n

   Please save this email.

   You will need your account ID to log in to the
   ".PROJECT." web site."

   );
}
```

Por esta otra, del mismo nombre:

```
function send_auth_email($email_addr, $auth) {

     ini_set("SMTP", "nyx.cti.espol.edu.ec");

     ini_set("sendmail_from","admin@supercomp.cti.espol.edu
     .ec");
```

```
    mail($email_addr, "Confirmación de nueva cuenta",
    "Este correo confirma la creación de tu cuenta en
    ".PROJECT."\n
    Web site:          ".MASTER_URL."\n
    Tu Identificador:   $auth\n

    Te recomendamos que guardes este correo.
    Necesitas este Identificador para ingresar al Sitio
    Web de ".PROJECT.".","From: Administrador de Super
    Computador ESPOL-CTI
    <admin@supercomp.cti.espol.edu.ec>");
}
```

Esto hará que el correo de verificación de la cuenta creada que el sistema envía para usuarios nuevos, sea mostrado en español y desde un remitente existente.

En el Apéndice C.1 se puede ver como el archivo `util.inc` debe quedar después de modificado.

A continuación se debe editar el archivo el archivo `countries.inc`, ubicado en el directorio `boinc/html/inc`, y ubicarse en la siguiente función:

```
function print_country_select($selected_country="None") {
   global $countries;
   //See if we can find the user's country and select it
   //as default:
   $gi = geoip_open("../inc/GeoIP.dat",GEOIP_STANDARD);
   $geoip_country                                          =
   geoip_country_name_by_addr($gi,$_SERVER["REMOTE_ADDR"]);
   geoip_close($gi);
   if ($selected_country=="") $selected_country="None";
   if ($selected_country=="None" and $geoip_country!=""){
        $selected_country=$geoip_country;
   }
   echo "selected: $selected_country\n";
```

```
    $numCountries = count($countries);
    for ($i=0; $i<$numCountries; $i++) {
         $country = $countries[$i];
         $selected  =  ($selected_country  ==  $country  ?
    "selected":"");
    echo             "<option             value=\"$country\"
    $selected>$country</option>\n";
     }
}
```

Se deben comentar desde la línea después del comentario hasta donde se declara la variable `numCountries`. De tal forma que la función quede de la siguiente manera:

```
function print_country_select($selected_country="None")
{
    global $countries;
    //See if we can find the user's country and select it as
    //default:
    /*$gi = geoip_open("../inc/GeoIP.dat",GEOIP_STANDARD);
     $geoip_country =
     geoip_country_name_by_addr($gi,$_SERVER["REMOTE_ADDR"]);
     geoip_close($gi);
     if ($selected_country=="") $selected_country="None";
     if ($selected_country=="None" and $geoip_country!=""){
        $selected_country=$geoip_country;
     }
     echo "selected: $selected_country\n";
    */
    $numCountries = count($countries);
    for ($i=0; $i<$numCountries; $i++) {
        $country = $countries[$i];
        $selected = ($selected_country == $country ?
    "selected":"");
    echo "<option value=\"$country\"
    $selected>$country</option>\n";
    }
}
```

En el Apéndice C.2 se puede ver como el archivo `countries.inc` debe quedar después de ser modificado.

5.2 Implementación de los clientes distribuidos

5.2.1 Instalación y configuración de los clientes

El sistema cliente fue instalado en cada una de las cinco computadoras cliente de prueba, tres máquinas de ellas pertenecientes al Centro de Tecnologías de Información (CTI) y dos máquinas particulares conectadas al Internet. En la Fig. 5.2.1-1, podemos apreciar la distribución de estas computadoras.

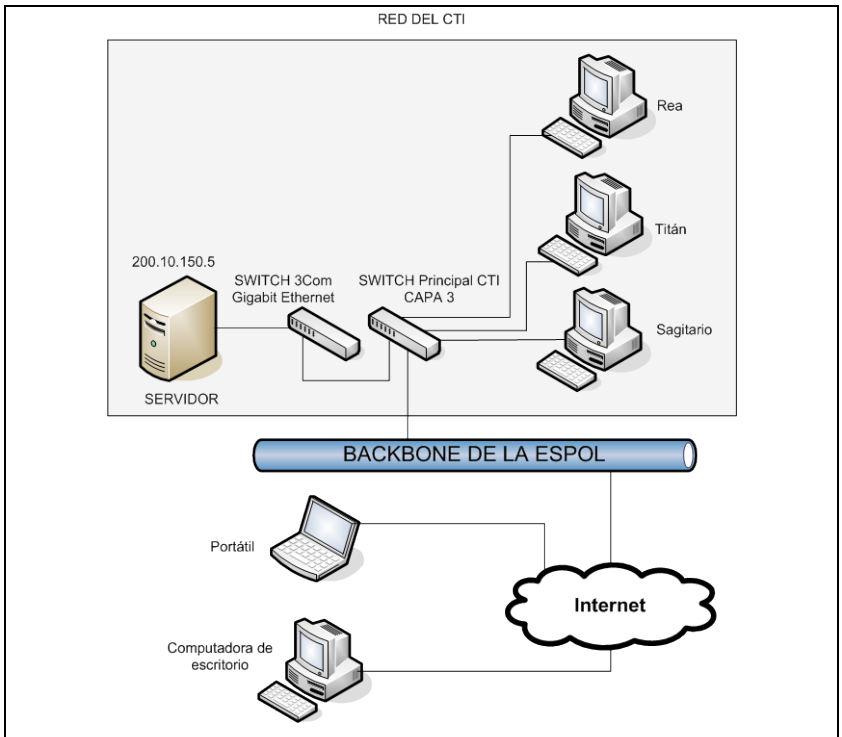


Figura 5.2.1-1. Esquema de Instalación de los clientes distribuidos.

Como podemos apreciar en la Fig. 5.2.1-1, las tres computadoras del CTI se comunican con el servidor a través de la red de área local y las dos restantes a través del Internet.

Todas estas computadoras son de arquitectura PC y tienen instalado el sistema operativo Windows XP; para más información de las características de cada computador, véase el Apéndice E.

La versión del cliente que se instaló fue la 3.05 para el sistema operativo Windows[1], el cual fue descargado desde el sitio Web del proyecto BOINC (http://boinc.berkeley.edu/download.php).

Para instalar el cliente, se debe hacer doble clic sobre el archivo, y luego seguir las instrucciones del instalador. Al final de la instalación se mostrará una ventana de configuración, véase Fig. 5.2.1-2; en la cual se deben seleccionar las siguientes opciones:

- Ejecutar BOINC al inicio.
- Establecer el protector de pantalla de BOINC por defecto.

[1] Esta versión es un instalador de Windows (boinc_3.05_windows_intelx86.exe).

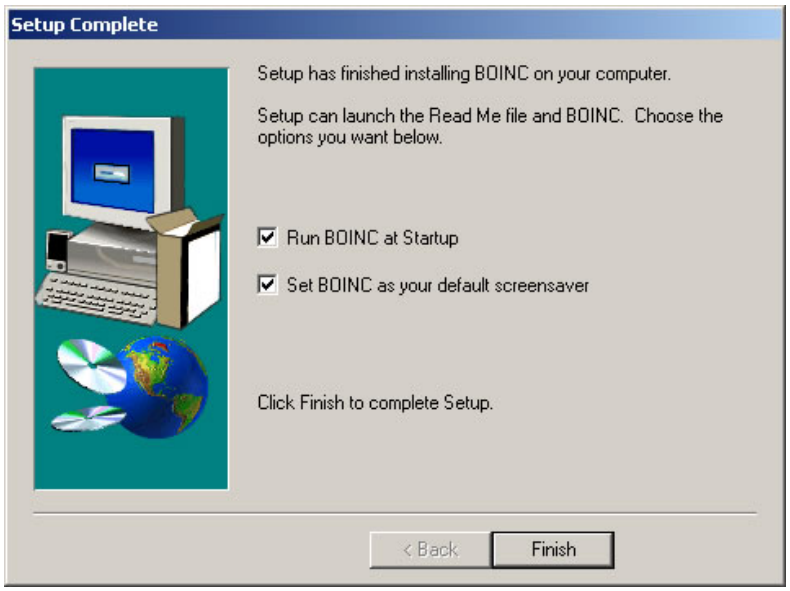


Figura 5.2.1-2. Configuración de cliente BOINC.

Para más información acerca de la instalación del cliente BOINC para Windows consulte el Anexo B.2.

Una vez instalado el cliente de BOINC, se mostrará un icono de BOINC en la barra de tareas, véase Fig. 5.2.1-3.

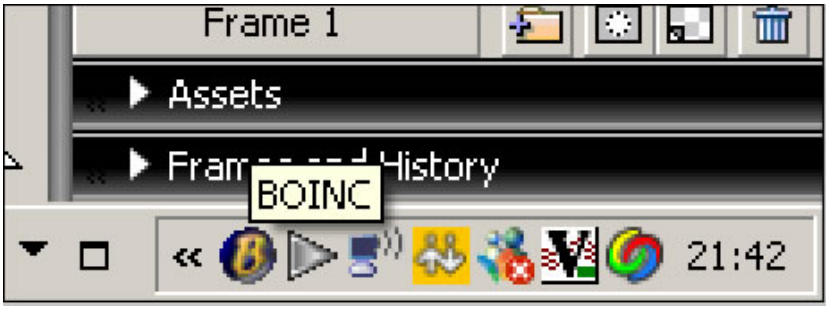


Figura 5.2.1-3. Icono de BOINC en la barra de tareas.

5.2.2 Cliente BOINC como servicio de Windows

Una posibilidad que BOINC nos ofrece es instalar el cliente como servicio de Windows, utilizando la versión de línea de comando del cliente de BOINC (CLI)[1].

Esta versión no se encuentra disponible para ser descargada desde Internet, si no que se debe compilar desde el código fuente usando Visual Studio .Net 2003.

Para probar esta alternativa de instalación; primero se compiló, y luego se instaló la versión CLI del cliente BOINC en la estación de trabajo Titán de la red del CTI.

Para obtener el ejecutable de esta versión del cliente BOINC; se debe descomprimir el archivo `boinc-cvs-2004-05-09`, luego abrir la solución `boinc.sln` ubicada en la carpeta `win_build` utilizando Visual Basic .Net 2003, y posteriormente generar el ejecutable del proyecto `boinc_cli`, véase Fig. 5.2.2-1.

---

[1] **boinc_cli.exe**, esta versión tipo consola es denominada CLI, por sus siglas en inglés Command Line Interface.

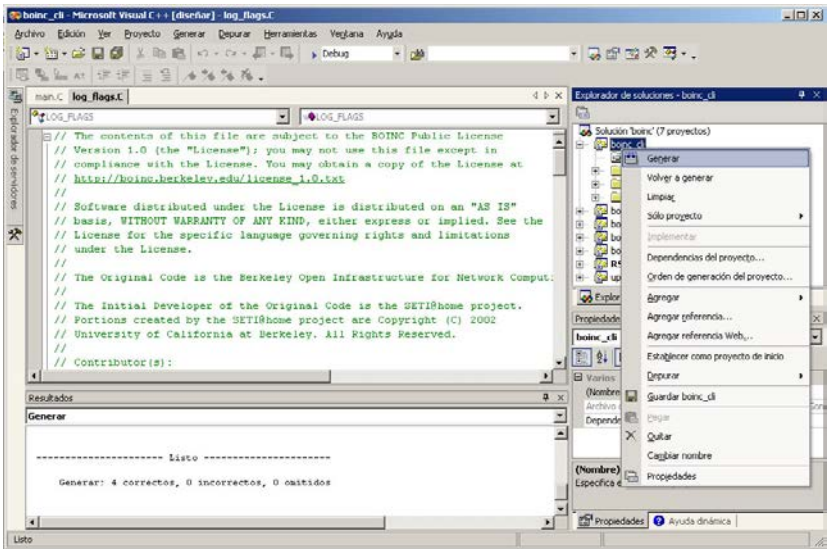


Figura 5.2.2-1. Compilación del cliente CLI de BOINC.

Esta versión del cliente BOINC debe ser instalada utilizando el símbolo del sistema de Windows, ejecutando el siguiente comando:

```
$boinc _cli -install
```

En la Fig. 5.2.2-2, se puede ver una pantalla de instalación del cliente BOINC como servicio.

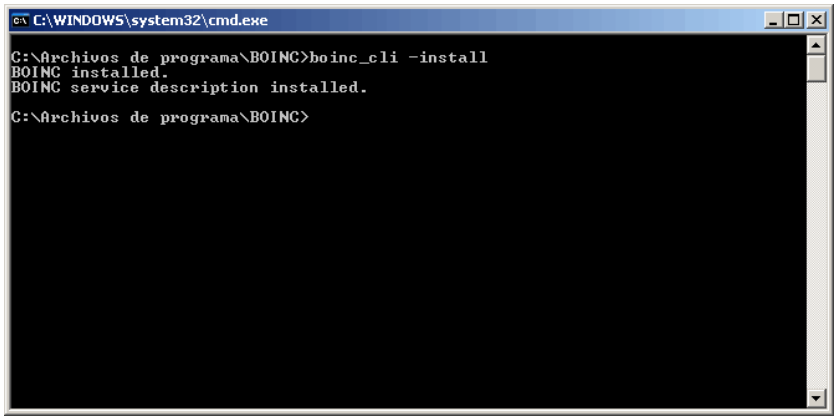


Figura 5.2.2-2. Instalación del cliente BOINC como servicio.

La ventaja de este tipo de instalación es que el cliente de BOINC se ejecuta en la estación de trabajo de manera transparente para el usuario.

5.3 Análisis y diseño de una aplicación distribuida

Luego de haber realizado la instalación y configuración de la plataforma BOINC y los clientes distribuidos, se procedió con el diseño de una aplicación distribuida la cual pueda ser ejecutada sobre nuestra plataforma BOINC, con la finalidad de poder evaluar el rendimiento y escalabilidad de nuestro sistema distribuido.

Esta aplicación distribuida, además de ser paralelizable, debía buscar la solución a un problema de gran desafío computacional, el cual una computadora común no lo pueda realizar o le tome demasiado tiempo.

Por otro lado, tenemos que los sistemas de descifrado de claves son aplicaciones que demandan de un gran poder computacional. Uno de los algoritmos de encriptación más populares es el RSA, desarrollado por la compañía RSA LABS, pioneras en el desarrollo de proyectos en el campo de la criptografía [REF.75].

El algoritmo RSA se fundamenta en el hecho de que la factorización de números primos muy grandes es muy complicada. Consiste de un sistema de cifrado de llave pública, el cual permite encriptación y firma digital [REF.75].

El algoritmo RSA funciona de la siguiente manera:

1. Se toman dos números primos muy grandes, llamados **p** y **q**, se calcula su producto **n=p*q**.
2. Se elige luego un numero, ***e***, el cual tiene que ser menor a **n** y relativamente primo a **(p-1)*(q-1)**, lo cual significa que **e** y **(p-1)*(q-1)** no tienen factores en común a excepción del numero 1.
3. Se busca otro número **d**, de tal modo que **(e*d - 1)** es divisible para **(p-1)*(q-1)**.
4. Los valores **e** y d son llamados como los exponentes públicos y privados respectivamente.
5. Formando los pares **(n, e)** la llave pública y el par **(n, d)** la llave privada [REF.75].

Actualmente es muy difícil obtener la clave privada d a partir de la clave pública (n, e). Sin embargo si se pudiera factorizar n en los valores p y q, entonces se podría obtener la clave privada d. El

sistema de seguridad RSA está basado en la dificultad de factorizar estos valores [REF.75].

Como vemos, el proceso de buscar factores primos es fácilmente paralelizable. Utilizando la técnica de paralelización de datos se puede dividir el valor n en rangos pequeños y asignar a cada cliente un rango de búsqueda diferente. Por todo esto, se desarrolló una aplicación distribuida que trate de descifrar una clave RSA de hasta 35 dígitos a través de la búsqueda de sus factores primos usando el método de fuerza bruta; es decir, mediante el proceso de divisiones sucesivas hasta encontrar uno de los factores primos; ésta aplicación distribuida se denominó **`rsadecrypt`**.

El objetivo general consiste en dividir el número **n** en rangos mucho más pequeños y manejables en términos de procesamiento, los cuales serán asignados a los clientes; a cada cliente se le asignará un rango de búsqueda diferente. El cliente realizará divisiones sucesivas sólo dentro de su rango asignado.

Para lograr esto se desarrolló:

1. Un componente encargado de dividir **n** en rangos más pequeños, y generar los archivos de entrada para la aplicación distribuida.

2. La aplicación distribuida que busca los factores primos dentro de un rango que le sea asignado.

El componente encargado de dividir n en rangos de búsqueda, lo llamamos **`rsasplitter`**, el cual debe cumplir con las siguientes funcionalidades:

1. Recibir una clave pública `(n,e)` a ser descifrada.
2. Crear los rangos de búsqueda desde 2 hasta la $\sqrt{n}$ [1].
3. Generar los archivos de entrada para la aplicación distribuida, los cuales almacenarán, entre otras cosas, la clave pública y el rango sobre el cual la aplicación distribuida debe realizar la búsqueda, véase Fig. 5.3-1.

n: Representa el valor n de la clave pública, un numero de N dígitos.

ninicial: Rango desde donde se comienza a hacer las divisiones sucesivas.

nfinal: Rango hasta donde se realizan las divisiones sucesivas.

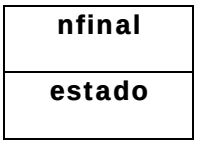
Figura 5.3-1. Estructura de los archivos de entrada.

[1] Según la Criba de Eratóstenes, los factores de un número n, para un número n = p x q tenemos que o bien p< $\sqrt{n}$ o bien q < $\sqrt{n}$ [REF.76].

Este componente es utilizado por el administrador del proyecto, en el cual la aplicación reside, para generar las unidades de trabajo. Estos archivos de entrada son generados como archivos de texto plano, y colocados en un directorio en el servidor.

Como vemos en la Fig. 5.3-3, la aplicación distribuida `rsadecrypt` que será enviada al cliente, debe procesar estos archivos de entrada provenientes del servidor como unidades de trabajo; leer los valores de `n`, `ninicial` y `nfinal`; para luego realizar divisiones sucesivas al valor de `n` comenzando desde `ninicial`. La unidad de trabajo termina de ser procesada cuando la aplicación distribuida llega a `nfinal,` o hasta que el residuo de la división es igual a cero, lo cual indica que se ha encontrado un factor primo. Este método de buscar factores a través de divisiones sucesivas se lo denomina “fuerza bruta”

Una vez que la aplicación distribuida procesa la unidad de trabajo, crea un archivo de salida a ser enviado al servidor como un resultado computacional obtenido por el cliente. Este archivo de salida, almacena el número hasta donde ha procesado, y una bandera de estado que indica si se ha encontrado o no la clave privada. En la Fig. 5.3-2 podemos ver la estructura de los archivos de salida.



nfinal: Hasta que numero llegó el cliente a procesar.

estado: El estado del resultado, 1 si logró encontrar la clave privada, 0 si no encontró.

Figura 5.3-2. Estructura de los archivos de salida.

El algoritmo que la aplicación distribuida utiliza para procesar las unidades de trabajo es el siguiente:

```
1.  Lee n, e, n_inicial y n_final de la unidad de
    trabajo.
2.  n_actual = n_inicial
3.  d = 0
4.  find_it = 0
5.  Si existe un estado previo
        a. Lee el registro del archivo de estado y lo
           asigna a n_actual.
6.  Si n Mod n_actual = 0
        a. find_it = 1
        b. factor_1 = n_actual
        c. factor_2 = n / n_actual
        d. Se calcula la clave privada a partir de
           factor_1 y factor_2 y es almacenada en d
        e. Ir a 12
7.  Si n_actual = n_final
        a. Ir a 12
8.  n_actual = n_actual +  1
9.  Si el protector de pantalla está activo
        a. Mostrar la parte gráfica
10. Si la aplicación es suspendida
        a. Ir a 11
11.Ir a 6
12.Se escribe en el archivo de estado un registro con el
    valor de n_actual.
    Ir a 12
13.Se crea el archivo de resultados y se almacenan los
    registros de n_actual y find_it
14.Fin
```

Otro aspecto importante de la aplicación es la capacidad de mostrar, en el protector de pantalla, gráficos que representan el procesamiento que el cliente esta ejecutando.

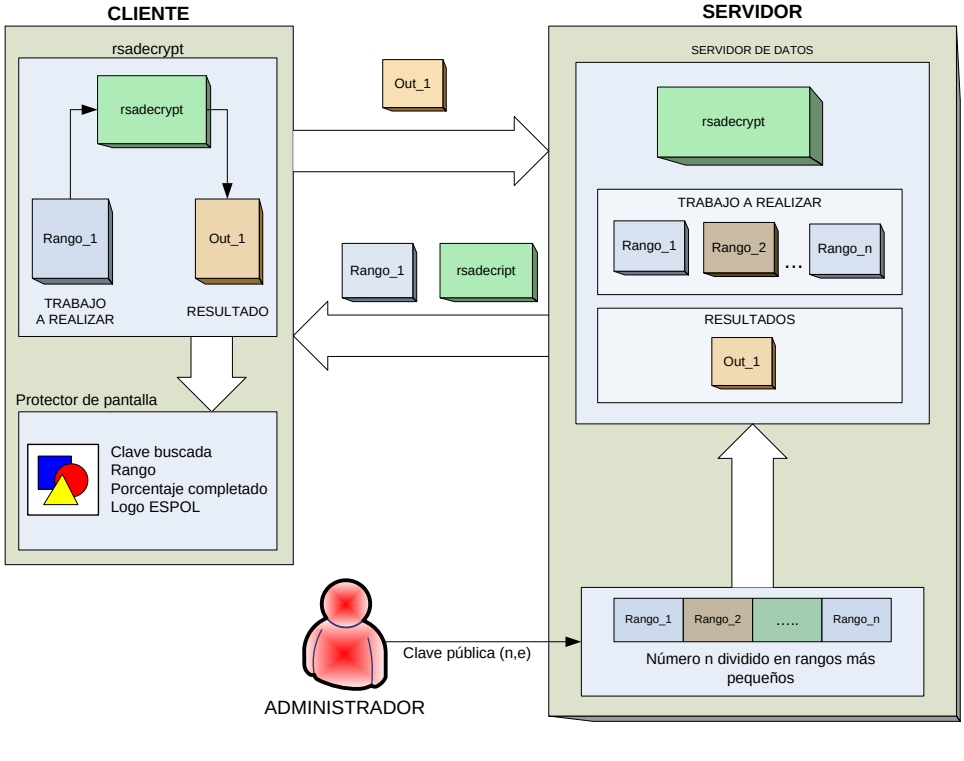


Figura 5.3-3. Interacción entre los componentes de la aplicación distribuida `rsadecrypt`.

5.4 Implementación de una aplicación distribuida

Tanto la aplicación distribuida `rsadecrypt`, como el componente para generar los archivos de entrada `rsasplitter`, fueron

implementados en Visual C++ Studio.NET 2003. En el desarrollo se utilizaron tanto el API de programación estándar de BOINC, así como también su API gráfico, ambos localizados en el instalador de la plataforma. Adicionalmente se utilizó la librería de programación gráfica openGL, y una librería para manipular y realizar operaciones con números grandes llamada LargeNumber, véase Apéndice F.

El API de programación estándar, es un conjunto de funciones implementadas en C++ que proveen funciones para el manejo de archivos, creación de puntos de chequeo y comunicación con el núcleo cliente; y la librería LargeNumber provee una nueva estructura de datos para representar números grandes y de las operaciones básicas de suma, resta, módulo, multiplicación y división.

Por las características de la plataforma BOINC, todas las operaciones de lectura/escritura de archivos deben utilizar la función `boinc_resolve_filename()` para hacer referencia al archivo que se desea abrir. Por ejemplo el archivo de entrada llamado `in` es leído de la siguiente manera:

```
boinc_resolve_filename("in",resolved_name,
                       sizeof(resolved_name));

in = fopen(resolved_name, "r");
```

La aplicación distribuida se comunica con el núcleo cliente en varias ocasiones, una de ellas es para indicar el porcentaje de trabajo que el cliente ha procesado, a través de la función `boinc_fraction_done`, la cual es usada de la siguiente manera:

```
preporcentaje=
 ((contador-iniciofijo)*CLargeNumber(1000)/(finfijo-
 iniciofijo)).ToString();

porcentaje=(double)(atof (&preporcentaje [0]))/1000;

boinc_fraction_done(porcentaje);
```

Adicionalmente el API de BOINC provee una función, que indica el momento de crear un punto de chequeo, llamada `boinc_time_to_checkpoint()`; ésta permite saber el momento de escribir el archivo de estado `cryp_state` que contiene el último número verificado por la aplicación distribuida antes de ser interrumpida. Esta función es usada de la siguiente manera:

```
bool flag = boinc_time_to_checkpoint();
if(flag){
  retval=do_checkpoint(&(contador.ToString()+char(13))[0]);
  if (retval) {
     fprintf(stderr, "Aplicacion: Check point de Decrypt ha
             fallado %d\n", retval);
     exit(1);
   }
   boinc_checkpoint_completed();
 }
```

De esta forma, cuando la aplicación es nuevamente iniciada, lo primero que hace es verificar si el archivo `cryp_state` existe, de tal forma que la búsqueda comience desde el número que este archivo contiene almacenado, véase el siguiente código:

```
boinc_resolve_filename(CHECKPOINT_FILE, resolved_name,
sizeof(resolved_name));
state = fopen(resolved_name, "r");
if (state) {
  fseek(state, 0L, SEEK_SET );
  fprintf(stderr, "Buscando el ultimo numero:" );
  fscanf(state, "%s", cadena_num_continuar);
  fprintf(stderr, "%s", cadena_num_continuar);
  fprintf(stderr, "%c", char(13));
  num_inicial = CLargeNumber(cadena_num_continuar);
  fclose(state);
}
```

Al finalizar el procesamiento de la unidad de trabajo se crea un archivo de salida, para que el núcleo cliente lo envíe al servidor como un resultado computacional, véase el siguiente código:

```
//Creo el archivo de salida o de resultado
boinc_resolve_filename("out", resolved_name,
sizeof(resolved_name));
retval = out.open(resolved_name, "w");

//Si no pudo crear el archivo se muestra un mensaje de
//error y se sale del programa con codigo 1

if (retval)
{
   fprintf(stderr, "Aplicacion: Error al crear el archivo
           de salida:\n");
   fprintf(stderr, "Nombre resuelto es %s, valor de retorno
           %d\n", resolved_name, retval);
   perror("open");
   exit(1);
}
```

```
out.puts(&numero_actual[0]);

out._putchar(13);

//Coloco el estado del procesamiento, 1 si encontró, 0 si
//no

if (encontrado){
   out.puts("1");
   fprintf(stderr, "APP: Se encontro la clave RSA!!!. Es:
           %s ", &numero_actual[0]);
}
else{
   out.puts("0");
   fprintf(stderr, "APP: No se econtro la clave RSA en esta
           unidad de trabajo");
}
retval = out.flush(); //escribir el archivo de salida
if (retval){
   fprintf(stderr, "APP: Ha ocurrido un error al guardar
       los datos en el archivo de salida: %d\n", retval);
   exit(1);
}
out.close();

fprintf(stderr, "APP: Decrypt finalizado. Se ha llegado
        hasta la clave %s \n", &contador.ToString()[0] );
```

Para la parte gráfica de la aplicación se utilizaron tanto el API gráfico de BOINC, así como las librerías de openGL para la generación de gráficos dinámicos y animaciones.

Se implementaron las funciones: `app_graphics_init()` y `void app_graphics_render(int xs, int ys, double time_of_day)`, la primera de ellas nos permite inicializar la parte gráfica de BOINC; y la segunda, dibujar el protector de pantalla:

```
void app_graphics_init() {
  //Incializa los graficos
       init_rsa_graphics();
}

void app_graphics_render(int xs, int ys, double
time_of_day) {
  //Evia a dibujar la animacion del protector de pantalla
  dibujar(time_of_day);
}
```

Como podemos ver en el código anterior, estas funciones a su vez llaman a otras funciones: `void init_rsa_graphics()` y `dibujar(time_of_day)`; ambas implementadas usando librerías `glut` y `gutils` de openGL. Todas estas funciones están implementadas en una librería llamada `windows_opengl.c`, véase Apéndice F.3.

Como vemos en la Fig. 5.4-1, el protector de pantalla implementado, muestra entre otras cosas, el valor de n, el rango en que el cliente debe realizar la búsqueda del factor primo, el número que está siendo verificado en ese instante, y el porcentaje de trabajo completado.

Figura 5.4-1. Protector de la aplicación rsadecrypt.

5.5 Creación del proyecto para la aplicación distribuida

Para poder utilizar y probar la aplicación distribuida desarrollada `rsadecrypt`, se creó una **aplicación de BOINC** dentro de un **proyecto** denominado **Proyecto Alpha**, en la plataforma BOINC instalada en el servidor.

Luego se configuraron los clientes distribuidos para que procesen trabajo del Proyecto Alpha y por consiguiente de la aplicación distribuida.

Para crear el proyecto se utilizó el script `make_project`, propio de la plataforma BOINC, localizado en el directorio `tools` de BOINC (`/boinc/tools`). De tal manera que el proyecto Alpha fue creado ejecutando la siguiente línea de comando:

```
$./boinc/tools/make_project --base /boinc --url_base
http://supercomp.espol.edu.ec/ alpha 'Supercomputador CTI,
Proyecto ALPHA'
```

Como podemos ver, este script, crea un proyecto con URL `http://supercomp.espol.edu.ec/alpha`; identificado como "Supercomputador CTI, Proyecto Alpha", cuyos archivos y directorios están localizados en `/boinc/projects/alpha`.

Este script, al ser ejecutado, realiza los siguientes pasos:

- Crea el directorio del proyecto y sus subdirectorios.
- Crea las llaves de encriptación del Proyecto, si es necesario.
- Crea e inicializa la base de datos MySQL.
- Copia los archivos fuentes y ejecutables.
- Genera el archivo de configuración del Proyecto (`config.xml`).

Como se aprecia en la Fig. 5.1-1, una vez ejecutado el script `make_project`, se muestran una serie de pasos que deben ser realizados para finalizar el proceso de instalación del proyecto.

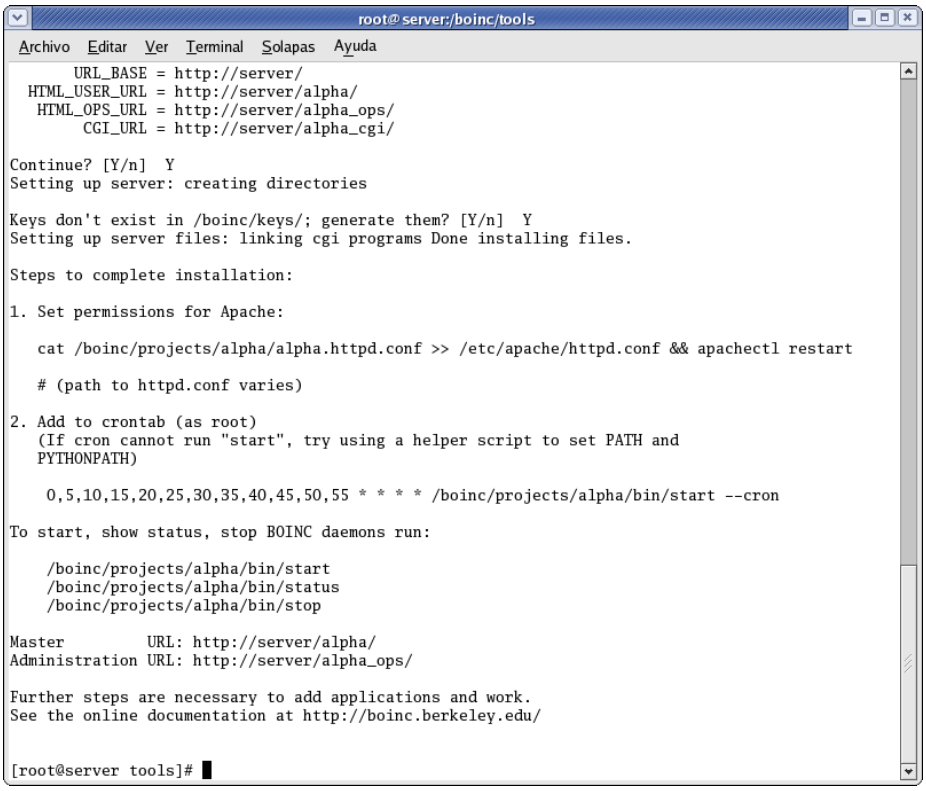


Figura 5.5-1. Instalación del Proyecto ALPHA.

Siguiendo esta secuencia de pasos, lo primero que se realizó fue la configuración del servidor Apache para permitir el acceso Web a los directorios `/html` y `/cgi-bin` del directorio del proyecto Alpha. De esta forma, se agregaron las líneas de código ubicadas en `/boinc/projects/alpha.httpd.conf` en el archivo de

configuración Apache `/etc/httpd/conf/httpd.conf`, ejecutando la siguiente línea de código:

```
$cat /boinc/projects/alpha/alpha.httpd.conf >>
/etc/httpd/conf/httpd.conf && service httpd restart
```

El segundo y último paso realizado dentro de la secuencia, fue la configuración del **cron**[1] de tal forma que ejecute el script `/boinc/projects/alpha/bin/star` cada cinco minutos de manera automática. Para hacer esto, primero se debe abrir el programador de tareas `crontab`, muy parecido al editor **vi**[2], ejecutando el siguiente comando:

```
$crontab –e
```

Una vez dentro del programador de tareas, se debe agregar el siguiente script:

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * *
/boinc/projects/alpha/bin/start –cron
```

[1] Utilidad de sistema que sirve para lanzar procesos con una periodicidad determinada, como por ejemplo copias de seguridad u otro tipo de procesos que deben ser lanzados de forma desatendida.

[2] Editor de texto elemental de Linux.

Luego se procede a guardar y salir del programador de tareas, utilizando los mismos comandos de **vi**.

Después de instalar el proyecto se realizaron ciertas configuraciones. Una de ellas fue en el servidor Web Apache; en primer lugar se lo configuró de tal forma que manipule los documentos de tipo MIME[1] como binarios, para lograr esto se debe editar el archivo de configuración de Apache `/etc/httpd/conf/httpd.conf` en la siguiente línea:

```
DefaultType plain/text
```

De tal forma que quede de la siguiente manera:

```
DefaultType application/octet-stream
```

En segundo lugar, se lo configuró para que el sitio Web por defecto sea el del proyecto Alpha. Para lograr esto, primero se debe comentar todas las líneas del archivo `/etc/http/conf.d/welcome.conf`, de tal forma que quede de la siguiente manera:

[1] MIME, acrónimo de Multipurpose Internet Mail Extensions, es una especificación para dar formato a mensajes no-ASCII, de tal forma que puedan ser enviados a través de Internet.

```
#
# This configuration file enables the default "Welcome"
# page if there is no default index page present for
# the root URL.  To disable the Welcome page, comment
# out all the lines below.
#
#<LocationMatch "^/+$">
#    Options -Indexes
#    ErrorDocument 403 /error/noindex.html
#</LocationMatch>
```

A continuación se debe crear un archivo llamado `index.html`, en el directorio `/var/www/html`, y escribir el siguiente código:

```
<html>
  <head>
    <script>
      location.href ="http://200.10.150.5/alpha";
    </script>
  </head>
</html>
```

Por último, se debe reiniciar el servicio Web de Apache, ejecutando la siguiente línea de comando:

```
$service iptables restart
```

Esta última configuración hará que la página Web por defecto del servidor Web sea la del Proyecto Alpha.

A continuación, se configuró el proyecto, editando el archivo `html/project/project.inc`, de tal forma que quede de la siguiente forma:

```
define("PROJECT", "Supercomputador CTI, Proyecto ALPHA");
define("MASTER_URL",
"http://supercomp.espol.edu.ec/aplha");
define("URL_BASE", "");
define("COPYRIGHT_HOLDER", "CTI, Centro de Tecnologias de
Informacion");
define("SYS_ADMIN_EMAIL",”kjdaskdjaskdjkl”);
```

En la Fig. 5.5-2 podemos ver la pantalla inicial del Proyecto Alpha.

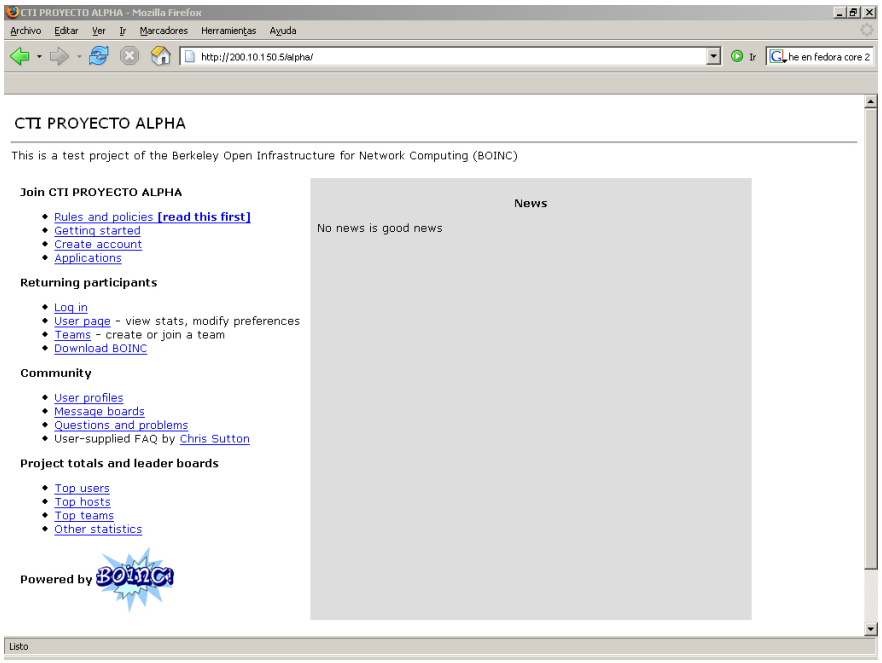


Figura 5.5-2. Pantalla inicial del Proyecto Alpha.

A continuación se agregó el subdominio **`supercomp.cti.espol.edu.ec`** en el archivo de zonas del DNS

de la ESPOL, de tal forma que los usuarios se conecten al servidor a través de `supercomp.cti.espol.edu.ec` y no a través de la IP.

Luego de esto, se debe registrar la aplicación `rsadecrypt` en el proyecto ALPHA; de esta forma, los clientes registrados en el proyecto procesarán trabajo para la aplicación. Para registrar la aplicación, debemos realizar lo siguiente:

1. Registrar la plataforma sobre la cual va a trabajar, en este caso Windows, el cual se lo identifica como **`windows_intelx86`**. Para agregar la plataforma se debe usar el comando `add` ubicado en la carpeta `/bin` del directorio del proyecto, de la siguiente manera

```
$./add platform --name=Windows_intelx86
                --user_friendly_name="Windows"
```

2. Registrar la aplicación, usando nuevamente el comando `add`, pero con parámetros diferentes, de la siguiente forma:

```
$./add app --name=rsadecrypt
           --user_friendly_name="Descifrador de Claves RSA"
```

A continuación se deben agregar al proyecto los archivos (ejecutables, librerías adicionales, entre otros) referentes a la

aplicación distribuida `rsadecrypt`. Estos archivos son agregados como una versión de la aplicación registrada. El nombre del archivo binario (ejecutable) de la aplicación debe tener el siguiente formato: **`nombreapp_version_plataforma`**. Se debe tomar en cuenta, que la versión debe ser siempre mayor o igual a la plataforma de BOINC[1] instalada en el servidor. En este caso el ejecutable tiene el siguiente nombre: **`rsadecrypt_3.06_windows_intelx86.exe.`**

Para agregar los archivos, se debe crear una carpeta con el nombre de la aplicación en el directorio /app del proyecto, y ejecutar el comando `update_versions` localizado en el directorio `/bin` del proyecto. En este caso, se creó la carpeta `rsadecrypt` dentro del directorio /app del proyecto ALPHA, en donde se copiaron los archivos: `rsa_decrypt_3.06_windows_intelx86.exe`, `logoespol.bmp` y `glut32.dll`. Adicionalmente se creó la carpeta `boinc` en este mismo directorio, en donde se agregó el cliente BOINC para Windows `boinc_3.05_windows_intelx86.exe`, con el objetivo de proporcionar el cliente BOINC a través del Internet.

Luego se ejecutó el siguiente comando:

```
$./update_versions
```

[1] Como se indicó en la sección 5.1.4, la versión de la plataforma BOINC instalada es la 3.04.

Se puede corroborar que la aplicación fue registrada y la versión agregada, visitando la sección “Applications” de la página principal del proyecto, véase Fig. 5.5-3.

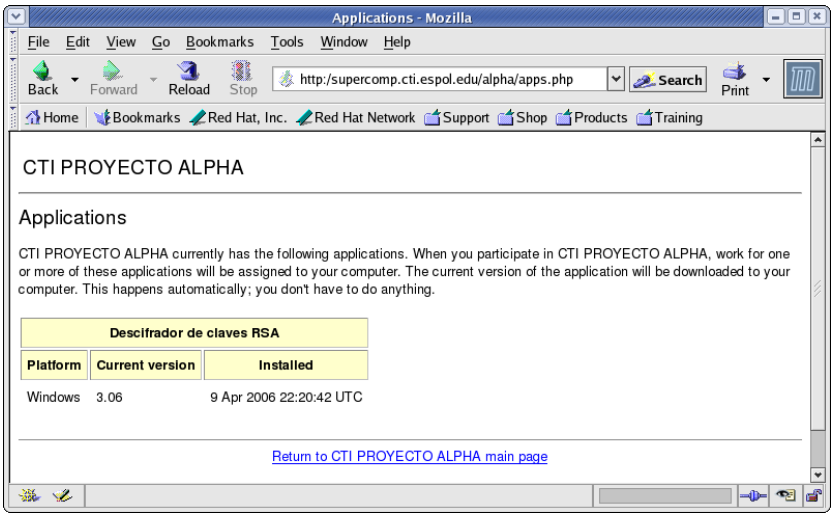


Figura 5.5-3. Pantalla de las aplicaciones.

Por último, se crearon las cuentas de usuario para cada una de las estaciones de trabajo y se configuraron los clientes para que procesen trabajo para el cluster. Para crear las cuentas de usuario se debe ir a la dirección Web de administración del proyecto: **`http://supercomp.cti.espol.edu.ec/alpha_ops,`** y seleccionar la opción de “Create Account” (Fig. 5.5-4).

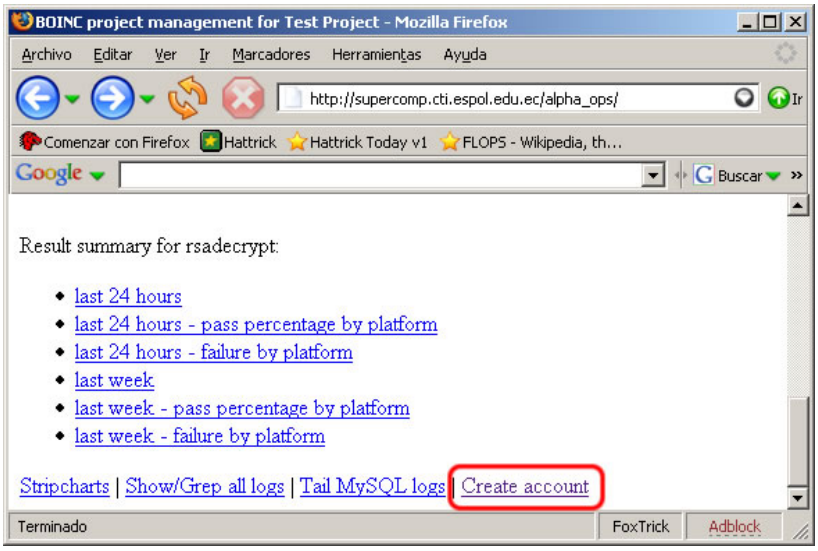


Figura 5.5-4. Página de administración del Proyecto Alpha.

Luego creamos cada una de las cuentas de usuario, colocando el nombre del usuario y la dirección de correo[1] (Fig. 5.5-5).

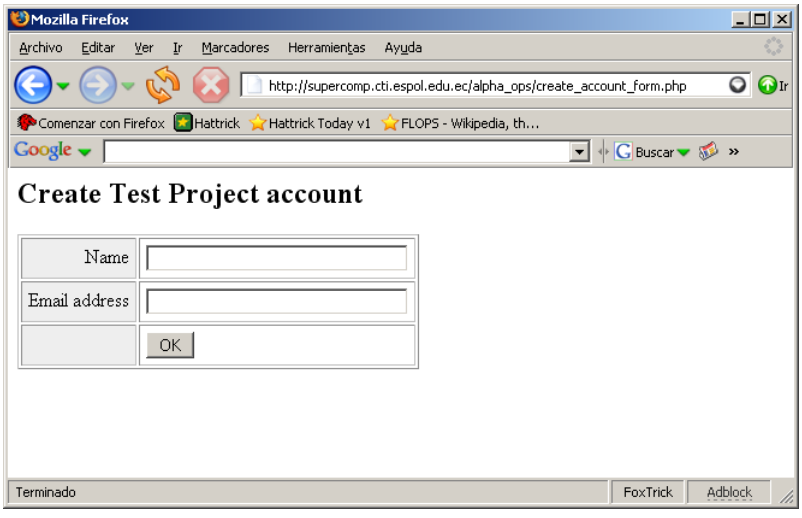


Figura 5.5-5. Registro de cuentas de usuario del Proyecto Alpha.

[1] En nuestro caso hemos ingresado una cuenta de correo ficticia para fines de pruebas.

Una vez que se haya realizado el registro de las 5 cuentas de usuario para cada una de las computadoras cliente; en la página de administración del proyecto, se debe seleccionar la opción “Users”, con el fin de obtener el código de autenticación (“Authenticator”) asignado a cada uno de los usuarios creados, el cual servirá para poder configurar los clientes (Fig. 5.5-6).

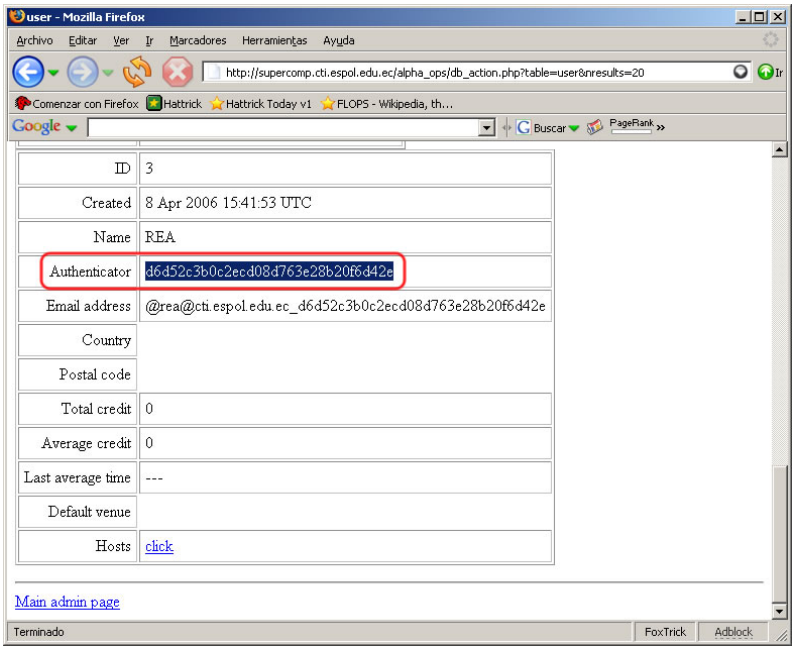


Figura 5.5-6. Código de autenticación del usuario.

Primero se debe configurar las preferencias de las cuentas de usuario creadas, para que los clientes inicien el procesamiento de datos luego de 6 segundos de inactividad del CPU. Para esto se debe iniciar sesión utilizando el código de autenticación del usuario

en la página principal del proyecto y editar las preferencias generales del usuario, véase Fig. 5.5-7. Repitiendo este procedimiento para cada una de las cuentas de usuario.

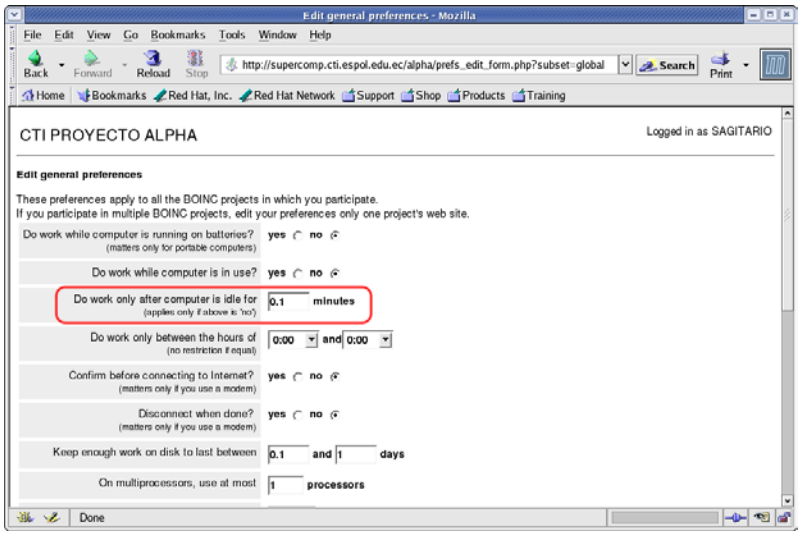


Figura 5.5-7. Preferencias generales del usuario.

Para configurar las computadoras cliente, se debe abrir el cliente BOINC, y seleccionar en el menú “Settings” la opción “Atach to Project…”, lo que mostrará una ventana de diálogo (Fig. 5.5-8), en donde se debe colocar la dirección Web del proyecto y el código de autenticación del usuario correspondiente a la estación cliente.

Este mismo procedimiento deberá ser ejecutado en cada una de las estaciones de trabajo que van a formar parte del cluster de computadoras.

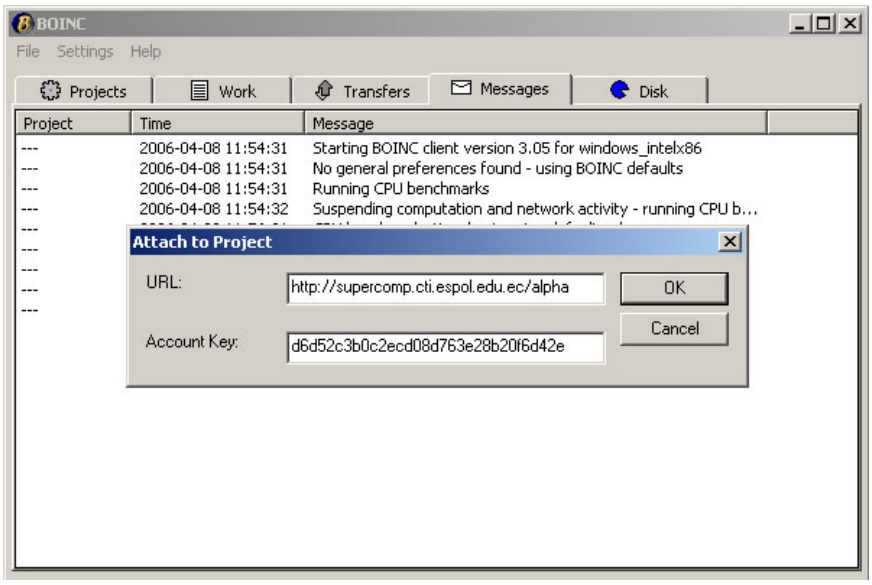


Figura 5.5-8. Ventana de configuración del cliente.

CAPÍTULO 6

6 PRUEBAS Y RESULTADOS OBTENIDOS

En el sexto capítulo se realiza una descripción de las pruebas de funcionamiento usando la aplicación distribuida desarrollada. Se muestran resultados de las mediciones realizadas en el sistema: de rendimiento, escalabilidad, aceleración (speedup), entre otras; todo esto para poder hacer estimaciones y proyecciones del poder computacional. Además se muestran datos comparativos (en tiempo de procesamiento en el descifrado de una clave RSA) entre el sistema distribuido de procesamiento desarrollado, y sistemas tradicionales no distribuidos.

6.1 Pruebas de funcionamiento

Para poder comprobar el funcionamiento del sistema distribuido de procesamiento, se utilizó la aplicación distribuida para descifrado de claves RSA, citada en las secciones 5.3 y 5.4, para encontrar los factores primos a partir del valor n de la clave pública.

En primer lugar se crearon los archivos de entrada[1] utilizando el módulo de creación de trabajo **`rsasplitter`**, de tal forma que el factor primo se encuentre en el último rango de datos a ser procesado, para detalles acerca de los archivos de entrada creados, véase Apéndice G. Estos archivos fueron enviados al servidor para poder generar las unidades de trabajo respectivas.

Para poder generar las unidades de trabajo, se deben crear las plantillas[2] tanto para las unidades de trabajo, como para los resultados computacionales. En este caso las plantillas fueron creadas en la carpeta `/templates` dentro del directorio del proyecto ALPHA.

---

[1] Estos archivos de entrada corresponderán a las unidades de trabajo de BOINC.

[2] Archivo XML que contiene información referente a la unidad de trabajo o al resultado computacional.

La plantilla creada para las unidades de trabajo para `rsadecrypt`, fue la siguiente:

```
<file_info>
  <number>0</number>
</file_info>
<workunit>
  <file_ref>
    <file_number>0</file_number>
    <open_name>in</open_name>
  </file_ref>
  <command_line>-brute_force</command_line>
  <delay_bounds>20000</delay_bounds>
  <target_nresults>1</target_nresults>
  <min_quorum>1</min_quorum>
  <rsc_fpops_est>1411200000000.000000</rsc_fpops_est>
 <rsc_fpops_bound>8467200000000.000000</rsc_fpops_bound>
  <rsc_memory_bound>70000000.000000</rsc_memory_bound>
  <rsc_disk_bound>100000000.000000</rsc_disk_bound>
</workunit>
```

Y la plantilla creada para los resultados computacionales fue la siguiente:

```
<file_info>
    <name><OUTFILE_0/></name>
    <generated_locally/>
    <upload_when_present/>
    <max_nbytes>100000</max_nbytes>
    <url><UPLOAD_URL/></url>
</file_info>
<result>
    <file_ref>
        <file_name><OUTFILE_0/></file_name>
        <open_name>out</open_name>
    </file_ref>
</result>
```

Una vez creadas las plantillas, se procede a crear las unidades de trabajo usando la herramienta `create_work` ubicada en la carpeta `/bin` del directorio del proyecto.

En este caso, para crear la primera unidad de trabajo se ejecutó la siguiente línea de comando:

```
$create_work -appname decrypt -wu_name rsadecrypt_wu_01
-wu_template
/boinc/projects/alpha/templates/DC_wu_template
-result_template
/boinc/projects/alpha/templates/DC_result_template
-db_name alpha -upload_url
http://200.10.150.5/alpha_cgi/file_upload_handler
-download_url http://200.10.150.5/alpha/download
-download_dir /boinc/projects/alpha/download
-keyfile /boinc/keys input_01
```

Las demás unidades de trabajo fueron creadas de la misma manera, variando el número correspondiente para cada unidad de trabajo.

Por último se levantaron los servicios correspondientes al Proyecto ALPHA, ejecutando la siguiente línea de comando:

```
$boinc/projects/alpha/bin/start
```

Luego de haber colocado trabajo para la aplicación `rsadecrypt`, se pudo observar como cada uno de los clientes se comunicó con el servidor de tareas, descargó las unidades de trabajo e inició el procesamiento. En la Fig. 6.1-1, se puede ver la aplicación cliente

descargando e iniciando el procesamiento de la unidad de trabajo que le fue asignada.

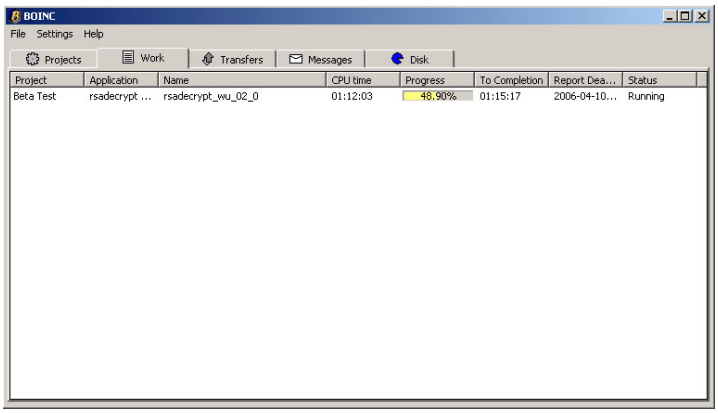


Figura 6.1-1. Cliente BOINC procesando trabajo.

También se realizaron pruebas de la parte gráfica de la aplicación distribuida. En la Fig. 6.1-2, se puede apreciar la computadora `Sagitario` procesando trabajo y su vez presentando la información gráfica en el protector de pantalla, referente al procesamiento que está realizando.

Figura 6.1-2. Estación de trabajo sagitario presentando el protector de pantalla de la aplicación `rsadecrypt`.

6.2 Medición de rendimiento

Para medir el rendimiento del sistema distribuido de procesamiento se utilizó como parámetro el tiempo (en minutos) necesario para la obtención del factor primo del número N, aumentando el número de computadoras clientes para la distribución del procesamiento. Para obtener los datos se realizaron las pruebas con 2 diferentes valores N (clave pública): de 12 y 14 dígitos.

La prueba realizada con un valor de N= 12 dígitos dio como resultado los siguientes valores:

Número de Computadoras	Num/seg	Tiempo en minutos
1	181	70
2	333	38
3	576	22
4	667	19
5	3168	4

Tabla 6.2-1 Tiempo de rendimiento para N=12.

Los datos utilizados para la obtención de estos resultados se encuentran en el Anexo G.1.

En la Fig. 6.2-1 se puede apreciar la mejora en el rendimiento del tiempo conforme se aumenta la cantidad de computadoras clientes al sistema distribuido de procesamiento.

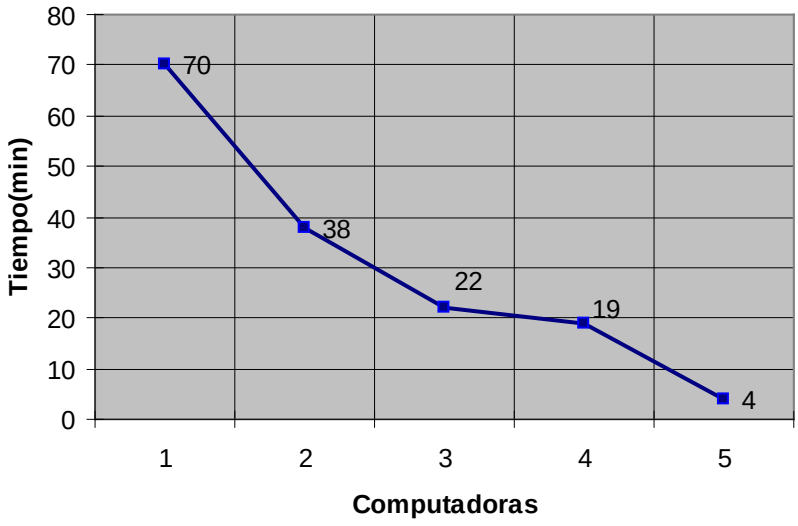


Figura 6.2-1. Gráfico de Computadoras vs Tiempo para N=12.

La prueba realizada con un valor de N= 14 dígitos dio como resultado los siguientes valores:

Número de Computadoras	Num/seg	Tiempo en minutos
1	345	260
2	642	140
3	1123	80
4	1284	70
5	4730	19

Tabla 6.2-2 Tiempo de rendimiento para N=14.

Los datos utilizados para la obtención de estos resultados se encuentran en el anexo G.2.

En la Fig. 6.2-2 se puede apreciar la mejora en el rendimiento del tiempo conforme se aumenta la cantidad de computadoras clientes al sistema distribuido de procesamiento.

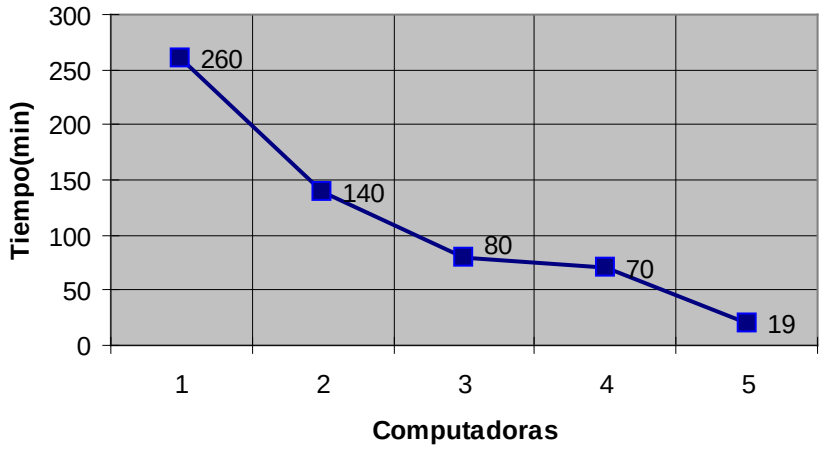


Figura 6.2-2. Gráfico de Computadoras vs. Tiempo para N=14.

En las pruebas realizadas, tanto para valores de N=12 y N=14 se puede apreciar un decremento notable en el tiempo de procesamiento utilizando 5 computadoras clientes. Esto se debe a que al procesarse todas las unidades de trabajo a la vez, el tiempo total de procesamiento es el tiempo en el cual una computadora procesa el rango en el cual se encuentra el factor primo que se está buscando.

Los resultados obtenidos reflejan claramente la escalabilidad del sistema distribuido de procesamiento, ya que al incrementar el número de computadoras, se disminuye el tiempo en obtener el resultado; esto es, el poder computacional está siendo incrementado a medida que se agregan más computadoras. Aunque hay que tomar en consideración, que el número de unidades a ser creadas debe ser mayor o igual al número de clientes en el sistema, de tal forma que se aprovechen los recursos de todas las computadoras presentes en el cluster.

6.3 Comparaciones con sistemas no distribuidos

En esta prueba se comparará el rendimiento del sistema distribuido de procesamiento con el rendimiento de una sola computadora en la

ejecución de la aplicación distribuida, pero esta vez midiendo la cantidad de números revisados por segundo.

La ejecución se realizó en 5 distintas computadoras independientes, para las cuales se obtuvieron los siguientes resultados de tiempo con valores N de 12 y 14 dígitos.

Computadora	**N=12**		**N=14**	
	Num/seg	**T (min)**	**Num/seg**	**T (min)**
1	186	68	360	249
2	186	68	393	258
3	218	64	413	230
4	232	60	457	222
5	101	138	201	505
Promedio	**192**	**80**	**374**	**292**

Tabla 6.3-1 Tiempos de rendimiento para una sola computadora.

Como se puede apreciar en la tabla 6.3-1, la cantidad promedio de números revisados es de 192 para un valor n de 12 dígitos y disminuye significativamente cuando la computadora es de características muy modestas, como por ejemplo la computadora #5.

Los datos de las características de las 5 computadoras clientes en las que se ejecutó la aplicación se encuentran en el Apéndice E.

Para poder comprobar que el sistema de procesamiento distribuido es mas rápido debemos calcular su aceleración (Speedup), que consiste en comparar los mejores tiempos de procesamiento para la búsqueda de la solución entre el sistema de procesamiento distribuido de procesamiento y el de una sola computadora (tabla 6.3-2), tenemos así que en promedio el sistema distribuido de procesamiento realiza la búsqueda de un factor primo de la clave RSA entre 10 y 14 veces más rápido que una computadora sola.

Mejor Rendimiento	**Aceleración (Speedup)**	
	N=12	**N=14**
1 Computadora	232	457
Sistema Distribuido	3168	4730
Número de computadoras		
1	0.78	0.75
2	1.44	1.41
3	2.48	2.46
4	2.88	2.81
5	13.65	10.35

Tabla 6.3-2 Comparación de rendimiento entre sistema de procesamiento distribuido y una computadora.

En la Fig. 6.3-1 podemos observar el comportamiento del Speedup del sistema de procesamiento distribuido, que para este caso en particular, incrementa en mayor proporción cuando se agregan más de 3 computadoras al sistema.

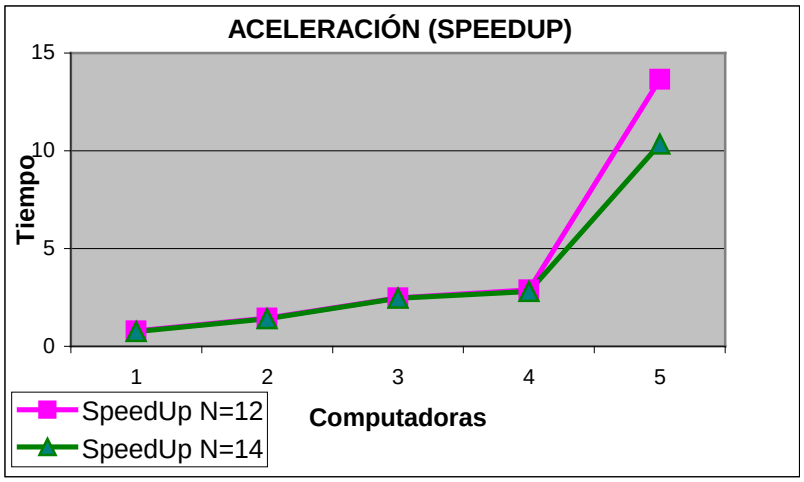


Figura 6.3-1. Gráfico de Speedup del Sistema de procesamiento distribuido.

Finalmente en la Fig. 6.3-2 se muestran los niveles de eficiencia del sistema de procesamiento distribuido en base a los resultados obtenidos, teniendo que para más de 4 computadoras clientes se tiene una eficiencia superior al 200%.

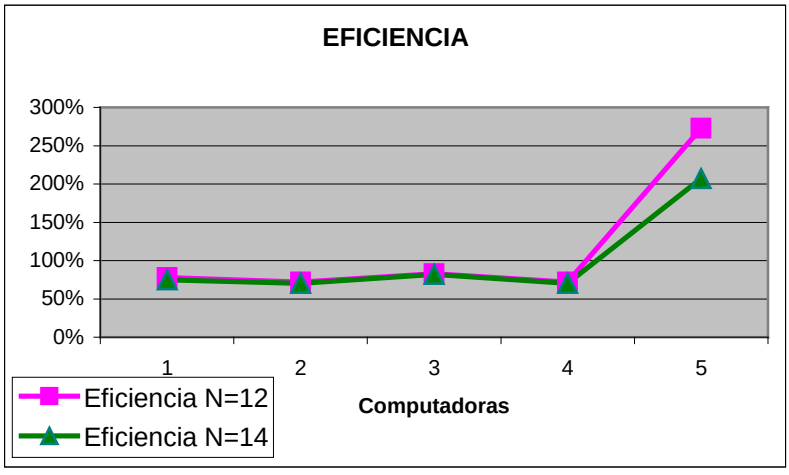


Figura 6.3-2. Gráfico de Eficiencia del Sistema de procesamiento distribuido.

Los datos utilizados para los gráficos de aceleración y eficiencia se encuentran en el anexo G.3.

En base a los resultados obtenidos para las pruebas realizadas, se puede concluir que para la ejecución de tareas que requieren de gran cantidad de procesamiento de datos es muy recomendable el uso de los sistemas distribuidos de procesamiento, debido a los altos niveles de escalabilidad, aceleración y eficiencia que presenta. La investigación científica, biotecnología, simulaciones, etc., son algunos campos en los cuales se puede aprovechar al máximo esta ventaja tecnológica, brindando así un mejor futuro para el desarrollo tecnológico del país.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

- La creación de un sistema distribuido de altas prestaciones utilizando los ciclos ociosos de los computadores personales, es una alternativa efectiva dentro de la ESPOL, con costos de instalación y operación bajos, y resultados bastante aceptables.
- Un sistema de procesamiento distribuido puede superar en poder de procesamiento a sistemas de súper cómputo convencionales.
- El sistema es completamente escalable, esto es, se puede incrementar el poder computacional con el simple hecho de aumentar el número de clientes distribuidos.
- Los problemas a ser resueltos usando nuestro sistema de procesamiento distribuido deben: ser paralelizables, soportar paralelización de datos y procesamiento independiente de datos.
- No todo problema que debe ser resuelto tiene la característica de ser paralelizable.
- La granularidad es un punto muy importante que hay que tomar en cuenta, ya que el número de unidades de trabajo creadas no debe ser menor al número de clientes en el sistema.

- La utilización de un protector de pantalla en el cliente, aunque da un mejor aspecto a la aplicación, hace que disminuya su capacidad de procesamiento.
- La aceleración además de depender del poder de procesamiento de los clientes, depende mucho del tiempo de comunicación entre las estaciones que forman el cluster.
- El ancho de banda que posea el servidor es otro aspecto importante, ya que, aunque que tengan muchos nodos clientes, si la comunicación con el servidor no es rápida, tanto las unidades de trabajo tomarán más tiempo en descargarse, así como los resultados tomarán más tiempo en ser enviados al servidor, lo que agregará más tiempo de procesamiento y disminuirá la aceleración.
- Podemos afirmar que se pueden aprovechar de manera efectiva los ciclos ociosos de los computadores, y quizás algún día nosotros mismos podamos donar nuestros ciclos ociosos, logrando de esta forma contribuir en la búsqueda de soluciones a grandes problemas.

Recomendaciones

- Promocionar a nivel interno y externo el servicio de computación distribuida desarrollado, para despertar el interés en el uso de esta nueva tecnología para resolver problemas de gran tamaño.
- Brindar la debida capacitación a los investigadores y profesores de la ESPOL para el uso de la computación distribuida, a través de cursos o talleres.
- Promover a la comunidad científica de la ESPOL e investigadores externos, el desarrollo de aplicaciones distribuidas, con el objetivo de encontrar nuevos y mejores resultados para el análisis extensivo de datos; como ejemplo simulaciones y predicción del clima, para tomar a tiempo medidas preventivas necesarias y poder así contrarrestar los efectos destructivos provocados por el calentamiento global del planeta y el fenómeno El Niño en nuestro país.
- Proponer la creación de un centro de alto rendimiento computacional al igual que lo hacen muchas universidades del mundo, en el cual se puedan aprovechar muchas de aquellas computadoras que hayan sido dadas de baja; y construir una poderosa computadora capaz de procesar grandes montos de información.

- Incorporar dentro de la materia “Sistemas Distribuidos” el concepto de Computación Voluntaria y desarrollar proyectos junto con los estudiantes para el desarrollo de aplicaciones distribuidas utilizando la plataforma BOINC.
- Invertir más tiempo y esfuerzo en el desarrollo de aplicación que optimicen el procesamiento, y no en mostrar demasiada información gráfica, ya que el rendimiento de la aplicación será disminuido. Es mejor que la información gráfica mostrada sea clara, concisa y sencilla, de tal forma que el cliente tome la mayor parte de su esfuerzo en realizar el cómputo y no en dibujar la parte gráfica.

APÉNDICES

A APÉNDICE A: SCRIPT DE CONFIGURACIÓN DEL FIREWALL

A.1 firewall.sh

```
#/bin/sh

# Variables

#Tarjeta de red eth0
if_principal=eth0

# Redes presentes
red_cti=200.10.150.0/24

# IP primaria
ip1=200.10.150.5/32

# IPs de host de confianza total
internet=0/0
lo=127.0.0.1/32

##Reglas por defecto
iptables -F
iptables -F INPUT
iptables -F OUTPUT
iptables -F FORWARD
iptables -F -t mangle
iptables -F -t nat
iptables -X
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

#### Reglas LOCALES
# Que se pueda conectar a si mismo
iptables -A INPUT -s $lo -d $lo -j ACCEPT
iptables -A OUTPUT -s $lo -d $lo -j ACCEPT

iptables -A INPUT -s $ip1 -d $ip1 -j ACCEPT
iptables -A OUTPUT -s $ip1 -d $ip1 -j ACCEPT

#### Reglas SERVIDOR-INTERNET por defecto
# Que pueda hacer ping
iptables -A OUTPUT -s $ip1 -d $internet -p icmp --icmp-type
echo-request -j ACCEPT
iptables -A INPUT -s $internet -d $ip1 -p icmp --icmp-type echo-
reply -j ACCEPT
```

```
# Resolucion de nombres (DNS)
iptables -A OUTPUT -s $ip1 -d $internet -p udp --dport 53 -j
ACCEPT

# Que pueda navegar (HTTP, HTTPS)
iptables -A OUTPUT -s $ip1 -d $internet -p tcp -m multiport --
dport 80,443 -j ACCEPT
iptables -A INPUT -s $internet -d $ip1 -p tcp -m multiport --
sport 80,443 -m state --state ESTABLISHED,RELATED -j ACCEPT

# Que pueda hacer ssh
iptables -A OUTPUT -s $ip1 -d $internet -p tcp --dport 22 -j
ACCEPT
iptables -A INPUT -s $internet -d $ip1 -p tcp --sport 22 -m
state --state ESTABLISHED,RELATED -j ACCEPT

# Enviar correo SMTP, PO3 e IMAP
iptables -A OUTPUT -s $ip1 -d $internet -p tcp --dport 25 -j
ACCEPT
iptables -A INPUT -s $internet -d $ip1 -p tcp --sport 25 -j
ACCEPT

# SSH
iptables -A INPUT -s $internet -d $ip1 -p tcp --dport 22 -j
ACCEPT
iptables -A OUTPUT -s $ip1 -d $internet -p tcp --sport 22 -j
ACCEPT

# HTTP para IP1
iptables -A INPUT -s $internet -d $ip1 -p tcp -m multiport --
dport 80,443 -j ACCEPT
iptables -A OUTPUT -s $ip1 -d $internet -p tcp -m multiport --
sport 80,443 -j ACCEPT

# Que le puedan hacer ping para IP1
iptables -A INPUT -s $internet -d $ip1 -p icmp --icmp-type echo-
request -j ACCEPT
iptables -A OUTPUT -s $ip1 -d $internet -p icmp --icmp-type
echo-reply -j ACCEPT

# MySQL (solo para red CTI)
iptables -A INPUT -s $red_cti -d $ip1 -p tcp --dport 3306 -j
ACCEPT
iptables -A OUTPUT -s $ip1 -d $red_cti -p tcp --sport 3306 -j
ACCEPT
```

B APÉNDICE B: MANUALES DE INSTALACIÓN

B.1 Instalación de Fedora Core 2

1. Utilizando el CD1 como disco de arranque, se enciende la computadora. Luego de que los controladores son cargados, se muestra la pantalla para seleccionar el modo de instalación, véase la Fig. B.1-1. Presione ENTER, para dar inicio a la instalación en modo gráfico.

Figura B.1-1. Pantalla de inicio de la instalación del sistema operativo Fedora Core 2.

2. Luego de que el instalador compruebe el contenido de cada uno de los CDs de instalación; se mostrará la pantalla de bienvenida de Anaconda, gestor de instalación en modo gráfico, véase Fig. B.1-2. Presionar Next para comenzar con la instalación.

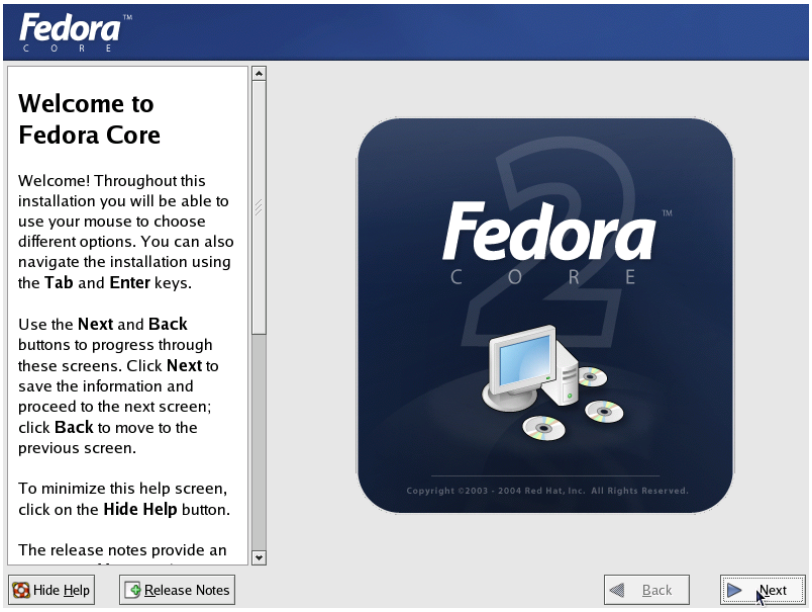


Figura B.1-2. Gestor de instalación de Fedora Core 4 en modo gráfico.

3. En las siguientes pantallas se debe seleccionar tanto el idioma de instalación, como el idioma del teclado. En nuestro caso seleccionamos español para ambos.
4. Después de haber configurado el idioma y teclado, se mostrará la pantalla de selección de Tipo de Instalación, véase Fig. B.1-3, en donde se debe seleccionar Personalizada, y luego presionar Siguiente**.**

Figura B.1-3. Pantalla de selección del Tipo de instalación.

5. En la pantalla de selección de Configuración del particionamiento de disco, véase Fig. B.1-4, aquí se debe seleccionar Partición manual con Disk Fruid, y luego presionar Siguiente.

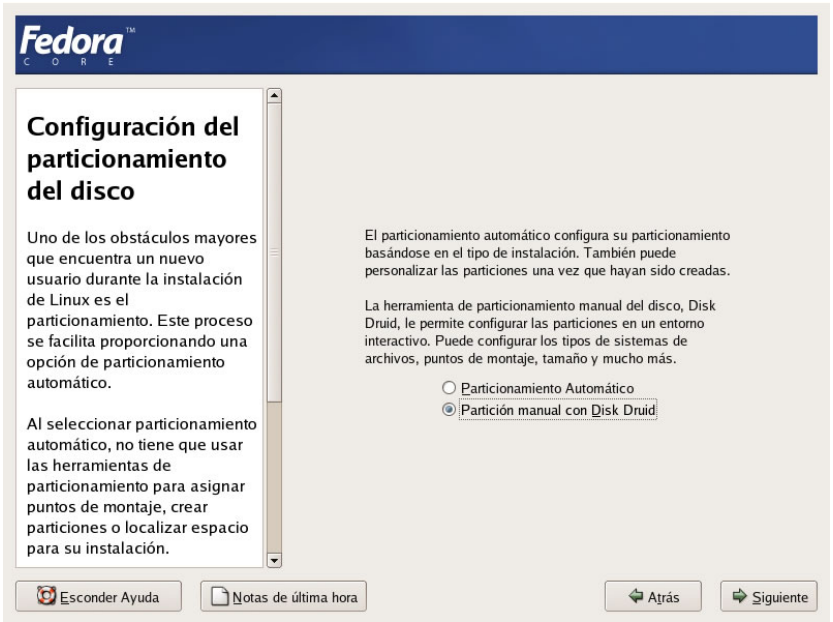


Figura B.1-4. Pantalla de selección de Configuración del particionamiento.

6. A continuación, se nos mostrará una ventana de aviso indicando que el particionamiento eliminará todos los datos presentes en la unidad. Presionar Si, indicando que estamos de acuerdo con éste proceso.
7. Configuramos el disco duro, especificando las siguientes particiones: Utilizar 100 MB para la partición de arranque (`/boot`); 3096 MB para la partición de intercambio (`swap`) y el resto de espacio de disco para la raíz del sistema (`/`). Luego de haber particionado el disco duro, presionar Siguiente. En la Fig. B.1-5, podemos observar como deben quedar las particiones del disco duro.

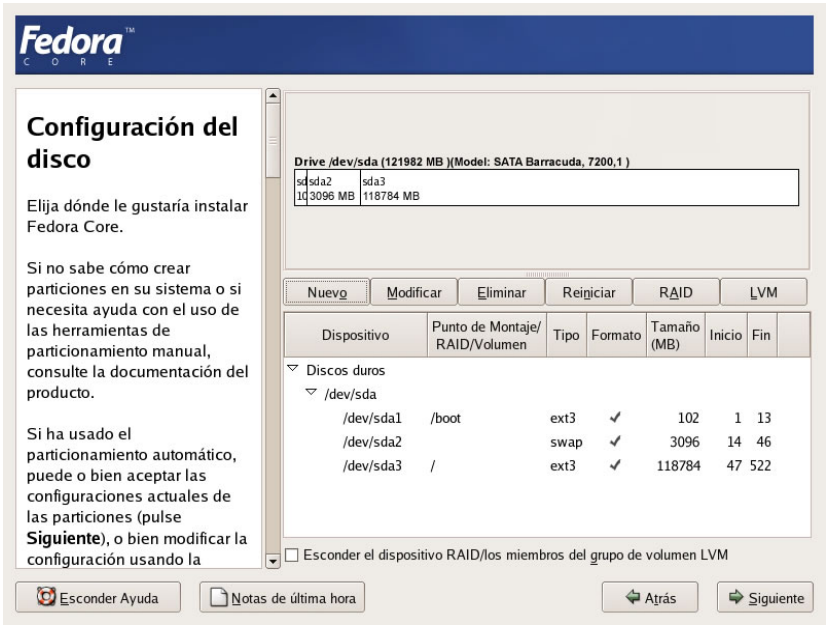


Figura B.1-5. Pantalla de configuración de las particiones del disco duro.

8. La pantalla se debe configurar el gestor de arranque, el cual nos ayudará a iniciar el sistema operativo. En esta pantalla no debemos realizar ningún cambio, sólo debemos presionar Siguiente.
9. A continuación debemos configurar la red, véase Fig. B.1-6. En esta parte debemos considerar que un requisito indispensable es que el servidor tenga un nombre de dominio, una IP pública, así como una puerta de enlace y servidores de dominio (DNS) que le permitan tener acceso al Internet.

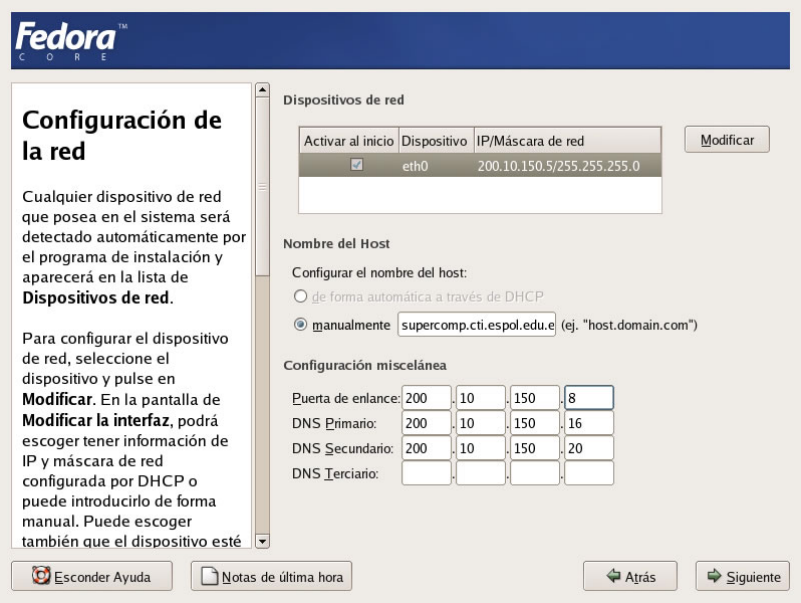


Figura B.1-6. Pantalla de configuración de la red.

En resumen, la red fue configurada de la siguiente manera:

Nombre del host	supercomp.cti.espol.edu.ec
IP	200.10.150.5
Máscara de red	255.255.255.0
Puerta de enlace	200.10.150.8
DNS Primario	200.10.150.16
DNS Secundario	200.10.150.20

Tabla B.1-1. Valores de configuración para la red en el servidor.

Una vez configurada la red, presionar Siguiente.

10. En la pantalla de configuración de cortafuegos, seleccionar la opción de Habilitar cortafuegos y presionar Siguiente.
11. A continuación se debe seleccionar la zona horaria, en nuestro caso América/Ecuador, y luego presionar Siguiente.
12. Asignar una contraseña para el administrador del equipo y presionar Siguiente.

Figura B.1-7. Pantalla de configuración de la contraseña para el root.

Debemos considerar en establecer una contraseña segura, difícil de adivinar, de no menos de ocho caracteres entre letras y números.

13. A continuación se muestra la ventana para seleccionar los grupos de paquetes que desea instalar, véase Fig. B.1-8.

Figura B.1-8. Pantalla de selección de grupo de paquetes para instalación.

En esta parte sólo los siguientes grupos de paquetes que deben estar seleccionados:

- **Sistema X Window.** Grupo de paquetes para utilizar la interfaz gráfica del sistema operativo.
- **Entorno de escritorio GNOME.** Interfaz gráfica de escritorio GNOME.
- **Herramientas de configuración del servidor.** Herramientas de configuración propias del sistema operativo.

- **Servidor Web.** Seleccionar los siguientes paquetes adicionales:
- **Servidor de Correo.** Paquetes para el servidor de correos.
- **Servidor FTP.** Paquetes para el servidor ftp y sftp.
- **Herramientas de desarrollo**. Principales herramientas de desarrollo de aplicaciones.
- **Desarrollo de software para X**. Paquetes para el desarrollo de aplicaciones en X Window.
- **Desarrollo de software anticuado**. Paquetes para soportar compatibilidad con sistemas anteriores.
- **Herramientas del sistema.** Seleccionar los siguientes paquetes adicionales:
- **Herramientas de administración**. Colección de herramientas gráficas de administración.

Una vez seleccionados cada uno de estos paquetes, presionar Siguiente.

14. Se muestra la pantalla de instalación de paquetes, donde se verifican las dependencias de los paquetes seleccionados, se da formato a cada una de las particiones creadas y se da inicio al proceso de instalación, véase la Fig. B.1-9.

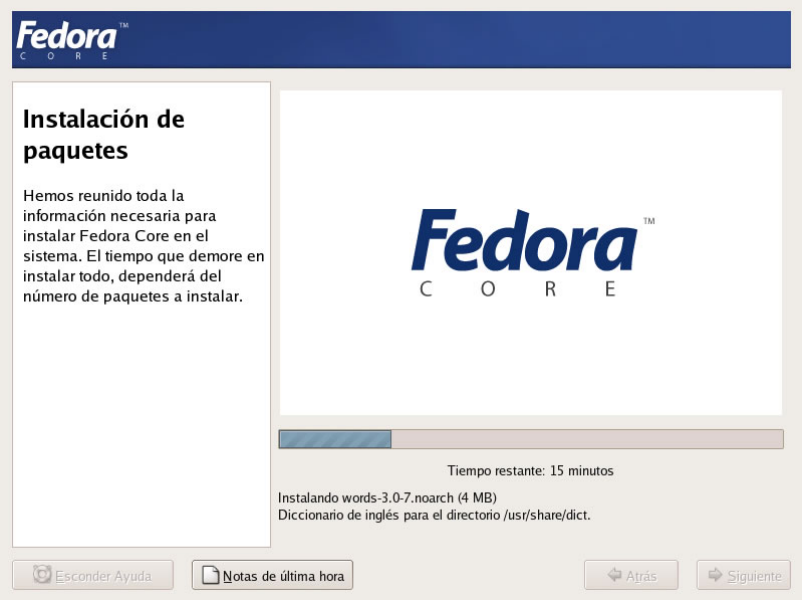


Figura B.1-9. Pantalla del proceso de instalación.

Cuando la instalación haya culminado se mostrará una pantalla que indicará que la instalación se ha realizado con éxito, véase Fig. B.1-10. Luego presionar Reiniciar.

Figura B.1-10. Pantalla de fin de la instalación.

15. Una vez que la computadora haya reiniciado, se mostrará una pantalla de bienvenida en donde deberá realizar algunas configuraciones adicionales al servidor, como fecha y hora, pantalla, usuario del sistema, y sonido.
16. Luego de haber realizado estas configuraciones adicionales, ingresar al sistema con el usuario administrador (`root`) y la contraseña que fue asignada.
17. A continuación se cargan todos los componentes del sistema operativo para luego mostrar el escritorio de Fedora Core 2, véase la Fig. B.1-11.

Figura B.1-11. Escritorio de Fedora Core 2.

B.2 Instalación del cliente BOINC para Windows

1. En la página principal del proyecto ALPHA, seleccionar la opcion “Download”, y hacer clic en descargar el cliente BOINC.
2. Ejecutamos la instalación del cliente BOINC, seleccionamos siguiente.

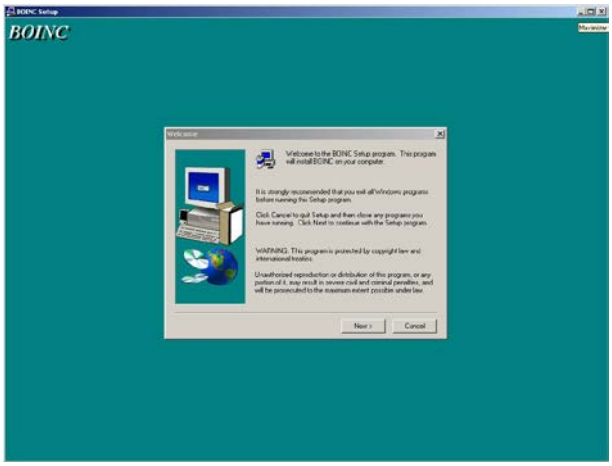


Figura B.2-1. Pantalla de inicio de la instalación

3. Luego de aceptar el contrato de licencia, se elije la ubicación de la carpeta de instalación y seleccionar siguiente.

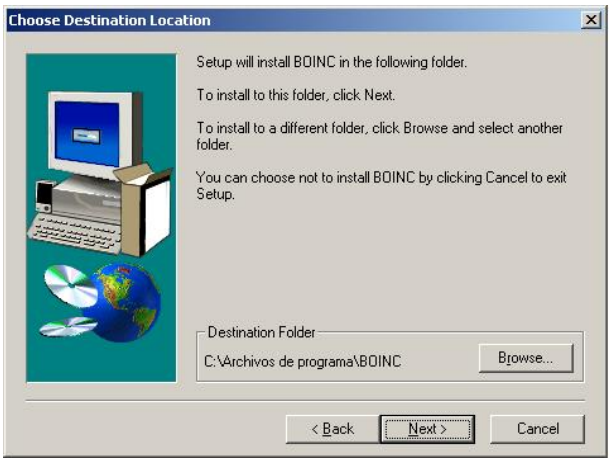


Figura B.2-2. Directorio de instalación

4. Seleccionar siguiente hasta llegar al cuadro final de la instalación, donde por defecto se ejecuta el cliente BOINC y se establece a BOINC como protector de pantalla.

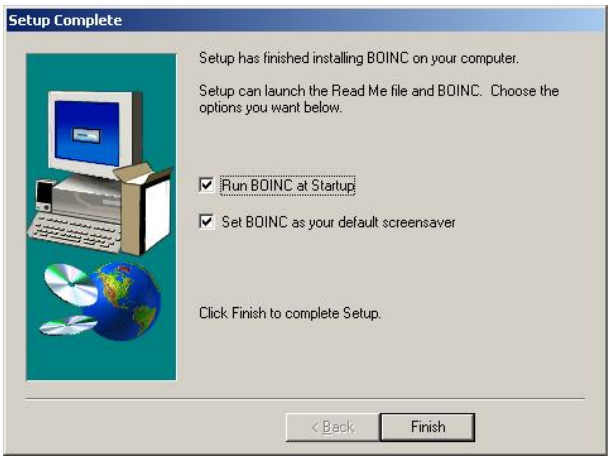


Figura B.2-3. Finalización de la instalación

5. Finalmente se ejecuta el cliente BOINC.

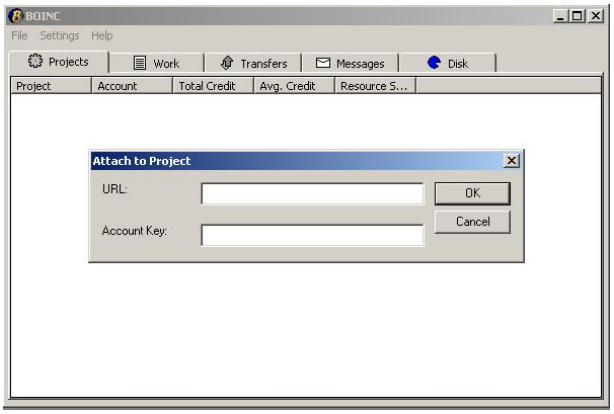


Figura B.2-4. Ventana principal del cliente BOINC

C APÉNDICE C: ARCHIVOS FUENTES DE BOINC

C.1 util.inc

```
<?php

require_once("../project/project.inc");
require_once("../inc/countries.inc");

// Sends the authenticator to the given email address
//
function send_auth_email($email_addr, $auth) {
ini_set("SMTP", "nyx.cti.espol.edu.ec");
ini_set("sendmail_from","admin@supercomp.cti.espol.edu.ec");

mail($email_addr, "Confirmación de nueva cuenta",
"Este correo confirma la creación de tu cuenta en ".PROJECT."\n
Web site:          ".MASTER_URL."\n
Tu Identificador:   $auth\n

Te recomendamos que guardes este correo.
Necesitas este Identificador para ingresar al Sitio Web de
".PROJECT.".","From: Administrador de Super Computador ESPOL-CTI
<admin@supercomp.cti.espol.edu.ec>");
}

// Initializes the session and returns the authenticator
// for the session (if any)
//
function init_session() {
    $url = parse_url(MASTER_URL);
    $path = $url['path'];
    if (strlen($path)) {
        session_set_cookie_params(0, $path);
    }
    session_start();
// NOTE: in PHP 4.1+, s/key_exists/array_key_exists/
    if (key_exists('authenticator', $_SESSION)) {
        return $_SESSION["authenticator"];
    } else {
        return NULL;
    }
}

// if not logged in, put up login form and exit
//
function require_login($user) {
    if (!$user) {
        print_login_form();
        exit();
    }
}
```

```
function get_user_from_auth($auth) {
    if ($auth) return lookup_user_auth($auth);
    return NULL;
}

function get_user_from_id($id) {
    if ($id) return lookup_user_id($id);
    return NULL;
}

function get_profile_from_userid($userid) {
    if ($userid) {
        $result = mysql_query("SELECT * FROM profile WHERE
userid = $userid");
        if ($result) {
            return mysql_fetch_assoc($result);
        }
    }
    return null;
}

function get_logged_in_user($must_be_logged_in=true) {
    $authenticator = init_session();
    if (!$authenticator) {
        $authenticator = $_COOKIE['auth'];
    }
    $user = get_user_from_auth($authenticator);
    if ($must_be_logged_in) {
        require_login($user);
    }
    return $user;
}

function show_login($user) {
    if ($user) {
        echo "Logoneado como %s.\n", $user->name;
        echo "<br><a href=login_form.php>Logoneado como alguien
mÃ¡s.</a>\n";
    } else {
        echo "No logoneado";
    }
}

// output a select form item with the given name,
// from a list of newline-delineated items from the text file.
// If $selection is provided, and if it matches one of the
entries in the file,
// it will be selected by default.
//
function show_combo_box($name, $filename, $selection=null) {
    if (!file_exists($filename)) {
        echo "ERROR: $filename no existe! No se puede crear el
combo box.<br>";
```

```
        exit();
    }
    echo "<select name=\"$name\">\n";

    $file = fopen($filename, "r");

    while ($line = trim(fgets($file, 1024))) {
        if ($line == $selection) {
            echo "<option SELECTED value=\"$line\">$line\n";
        } else {
            echo "<option value=\"$line\">$line\n";
        }
    }

    echo "</select>\n";
    fclose($file);
}

// write to the given fd if non-null; else echo
//
function write_fd($fd, $str) {
    if ($fd) {
        fwrite($fd, $str);
    } else {
        echo $str;
    }
}

function page_head($title, $user=null, $fd=null) {
    $styleSheet = URL_BASE . STYLESHEET;

    write_fd($fd,
        "<html><head><title>$title</title>
        <link rel=stylesheet type=text/css href=\"$styleSheet\">
        </head>
        <body bgcolor=ffffff>
        "
    );
    project_banner($user, $fd);
}

function page_tail($is_main=false, $fd=null) {
    write_fd($fd, "<br><hr noshade size=1><center>");
    if (!$is_main) {
        write_fd($fd, "<a href=".MASTER_URL.">Regresar a la
pÃ¡gina principal del  ".PROJECT."</a><br>\n");
    }

    // put your copyright notice etc. here

    write_fd($fd, "<br><br>Copyright &copy; 2004
".COPYRIGHT_HOLDER."</center>\n</body>\n</html>");
}
```

```
function db_error_page() {
    page_head("Error en la Base de Datos");
    echo "<h2>Error en la Base de Datos</h2>
        Un error en la base de datos ha ocurrido mientras se
maneja tu requerimiento.
        <br>Por favor, intenta en unos minutos.
        <br>Si el error persiste, reporta tu problema
        <a href=bug_report_form.php>Haciendo clic aquÃ</a>.
    ";
    page_tail();
}

function profile_error_page($str) {
    page_head("Profile error");
    echo "$str<br>\n";
    echo "<p>Haz clic en el botÃ²n <b>Regresar</b> de tu
navegador e intenta de nuevo.\n<p>\n";
    page_tail();
}

function date_str($x) {
    if ($x == 0) return "---";
    // return date("g:i A, l M j", $when);
    return strftime("%Y-%m-%d", $x);
}

function time_str($x) {
    if ($x == 0) return "---";
    //return strftime("%j %m-%d %H:%M:%S", $x)";
    return gmdate('j M Y G:i:s', $x) . " UTC";
}

function pretty_time_str($x) {
    return time_str($x);
}

function start_table($extra="") {
    echo "<table border=1 cellpadding=5 $extra>";
}

function start_table_noborder($width="100%") {
    echo "<table border=0 cellpadding=5 width=$width>";
}

function end_table() {
    echo "</table>\n";
}

function row1($x, $ncols=2) {
    echo "<tr><td class=heading bgcolor=cccccc
colspan=$ncols><b>$x</b></td></tr>\n";
}

function row2($x, $y) {
```

```
    if ($x=="") $x="<br>";
    if ($y=="") $y="<br>";
    echo "<tr><td class=fieldname bgcolor=eeeeee align=right
valign=top>$x</td><td valign=top><b>$y</b></td></tr>\n";
}
function row2_init($x, $y) {
    echo "<tr><td class=fieldname bgcolor=eeeeee width=40%
align=right valign=top>$x</td><td valign=top><b>$y\n";
}

function row2_plain($x, $y) {
    echo "<tr><td>$x</td><td>$y</td></tr>\n";
}

function row3($x, $y, $z) {
    echo "<tr><td width=30% valign=top
align=right>$x</td><td>$y</td><td>$z</td></tr>\n";
}

function rowify($string) {
    echo "<tr><td>$string</td></tr>";
}

function random_string() {
    return md5(uniqid(rand(), true));
}

function print_login_form_aux($next_url, $user) {
    echo "
        <form name=f method=post action=login_action.php>
        <input type=hidden name=next_url value='$next_url'>
    ";
    start_table();
    row1("Ingreso");
    row2("Tu Identificador:
        <br><font size=-2>
        Si tu no conoces tu identificador,
        <a href=get_passwd.php>haz clic aquÃ</a>.
        </font>",
        "<input name=authenticator size=40>"
    );
    row2("Recordar el Identificador en la computadora",
        "<input type=checkbox name=send_cookie>"
    );
    row2("",
        "<input type=submit value='Ingresar'>"
    );
    if ($user) {
        row1("Cerrar SesiÃ³n");
        row2("Tu estÃ¡s logoneado como $user->name",
            "<a href=logout.php>Cerrar SesiÃ³n</a>"
        );
    }
    end_table();
```

```
    echo "
        </form>
        <script>
            document.f.authenticator.focus();
        </script>
    ";
}

function print_login_form() {
    page_head("Logonear");
    echo "
        <h3>Logoneate al sitio</h3>
        Esta funcionalidad requiere que estÃ©s logoneado.
    ";
    $next_url = $_SERVER['REQUEST_URI'];
    print_login_form_aux($next_url, null);
    page_tail();
}

// Look for an element in a line of XML text
// If it's a single-tag element, and it's present, just return
the tag
//
function parse_element($xml, $tag) {
    $element = null;
    $x = strstr($xml, $tag);
    if ($x) {
        if (strstr($tag, "/>")) return $tag;
        $y = substr($x, strlen($tag));
        $n = strpos($y, "<");
        if ($n) {
            $element = substr($y, 0, $n);
        }
    }
    return $element;
}

if (!function_exists("file_get_contents")) {
function file_get_contents($path) {
    $x = "";
    $f = fopen($path, "r");
    if ($f) {
        while (!feof($f)) $x .= fread($f, 4096);
        fclose($f);
    }
    return $x;
}
}

// look for a particular element in the ../../config.xml file
//
function parse_config($tag) {
    $element = null;
    $buf = file_get_contents("../../config.xml");
```

```
    $element = parse_element($buf, $tag);
    return trim($element);
}

// Call this if for dynamic pages
//
function no_cache() {
    header ("Expires: Mon, 26 Jul 1997 05:00:00 GMT");    //
Date in the past
    header ("Last-Modified: " . gmdate("D, d M Y H:i:s") . "
GMT"); // always modified
    header ("Cache-Control: no-cache, must-revalidate");  //
HTTP/1.1
    header ("Pragma: no-cache");                          //
HTTP/1.0
}

// a valid email address is of the form A@B.C
// where A, B, C are nonempty,
// A and B don't contain @ or .,
// and C doesn't contain @
//
function is_valid_email_addr($addr) {
    $x = strstr($addr, "@");
    if (!$x) return false;
    if (strlen($x) == strlen($addr)) return false;
    $x = substr($x, 1);
    if (strstr($x, "@")) return false;
    $y = strstr($x, ".");
    if (!$y) return false;
    if (strlen($y) == strlen($x)) return false;
    if (strlen($y) == 1) return false;
    return true;
}

// A few functions relating to email-address munging
// A "munged" email address is of the form @X_Y,
// where X is a valid email address
// and Y is a random string not containing _.
// When an email address hasn't been validated yet, it's munged.
// (Used during account creation and email address changes)

function munge_email_addr($email, $string) {
    return "@".$email."_".$string;
}

// if email_addr is of the form @X_Y, split out the X and return
true.
// otherwise return false
//
function split_munged_email_addr($addr, $string, &$email) {
    if (substr($addr, 0, 1) != "@") return false;
    $x = strrchr($addr, "_");
    if (!$x) return false;
```

```
    $y = substr($x, 1);
    if ($y != $string) return false;
    $email = substr($addr, 1, strlen($addr)-strlen($x)-1);
    return true;
}

// Generates a standard set of links between associated multi-
page documents.
//  All linked files must be of the form "$filename_<page
number>.html".

function write_page_links($filename, $currPageNum, $numPages,
$descriptor) {
    fwrite($descriptor, "<p>Page $currPageNum of
$numPages</p>");

    $nextPageNum = $currPageNum + 1;
    $prevPageNum = $currPageNum - 1;

    // Make the 'previous' and 'next' page links as appropriate.
    if ($currPageNum > 1) {
        fwrite($descriptor, "<a href=$filename" . "_" .
$prevPageNum . ".html>Previous Page</a>");

        if ($currPageNum != $numPages) {
            fwrite($descriptor, " | ");
        }
    }
    if ($currPageNum != $numPages) {
        fwrite($descriptor, "<a href=$filename" . "_" .
$nextPageNum . ".html>Next Page</a>");
    }

    fwrite($descriptor, "<p>Jump to Page:\n");

    // Make the individual page links (or a bold non-link for
the current page).
    //
    for ($i = 1; $i <= $numPages; $i++) {
        if ($i != $currPageNum) {
            fwrite($descriptor, "<a href=$filename" . "_" . $i .
".html>$i</a>\n");
        } else {
            fwrite($descriptor, "<b>$i</b>\n");
        }
    }

}

// Generates a legal filename from a parameter string.

function get_legal_filename($name) {
    $name = ereg_replace(',', '', $name);
    return ereg_replace(' ', '_', $name);
```

```
}

// Returns a string containing as many words
// (being collections of characters separated by the character
$delimiter)
// as possible such that the total string length is <= $chars
characters long.
// If $ellipsis is true, then an ellipsis is added to any
sentence which
// is cut short.

function sub_sentence($sentence, $delimiter, $max_chars,
$ellipsis=false) {
    $words = explode($delimiter, $sentence);
    $total_chars = 0;
    $count = 0;
    $result = '';

    do {
        if ($count > 0) {
            $result = $result . ' ' . $words[$count];
        } else {
            $result = $result . $words[$count];
        }
        $total_chars += strlen($words[$count]) + 1;
        $count++;
    } while ($count < count($words) && ($total_chars +
strlen($words[$count])) <= $max_chars);

    if ($ellipsis && ($count < count($words))) {
        $result = $result . '...';
    }

    return $result;
}

function format_credit($cobblestones) {
    return sprintf("%.2f", $cobblestones);
}

function project_is_stopped() {
    return file_exists("../stop_servers");
}

function user_links($user, $dir='') {
    $x = "<a href=".$dir."show_user.php?userid=$user->id>$user-
>name</a>";
    if ($user->has_profile) {
        $x .= " <a href=".$dir."view_profile?userid=$user-
>id><img border=0 src=".$dir."head_20.png></a>";
    }
    return $x;
}
```

```
function host_link($hostid) {
    if ($hostid) {
        return "<a
href=show_host_detail.php?hostid=$hostid>$hostid</a>";
    } else {
        return "---";
    }
}

function news_item($date, $text) {
    echo "<b>$date</b>
        <br>
        $text
        <br><br>
    ";
}

?>
```

C.2 countries.inc

```
<?php
// $Id: countries.inc,v 1.4 2004/04/21 23:25:04 boincadm Exp $
// list of countries taken from http://www.cia.gov

require_once("../inc/geoip.inc");

$countries = array(
    "None",
    "International",
    "United States",
    "Afghanistan",
    "Albania",
    "Algeria",
    "American Samoa",
    "Andorra",
    "Angola",
    "Anguilla",
    "Antarctica",
    "Antigua and Barbuda",
    "Argentina",
    "Armenia",
    "Aruba",
    "Australia",
    "Austria",
    "Azerbaijan",
    "Bahamas, The",
    "Bahrain",
    "Bangladesh",
    "Barbados",
    "Belarus",
    "Belgium",
    "Belize",
    "Benin",
    "Bermuda",
```

```
"Bhutan",
"Bolivia",
"Bosnia and Herzegovina",
"Botswana",
"Brazil",
"British Virgin Islands",
"Brunei",
"Bulgaria",
"Burkina Faso",
"Burundi",
"Cambodia",
"Cameroon",
"Canada",
"Cape Verde",
"Cayman Islands",
"Central African Republic",
"Chad",
"Channel Islands",
"Chile",
"China",
"Colombia",
"Comoros",
"Congo, Democratic Republic of the",
"Congo, Republic of the",
"Cook Islands",
"Costa Rica",
"Cote d'Ivoire",
"Croatia",
"Cuba",
"Cyprus",
"Czech Republic",
"Denmark",
"Djibouti",
"Dominica",
"Dominican Republic",
"East Timor",
"Ecuador",
"Egypt",
"El Salvador",
"Equatorial Guinea",
"Eritrea",
"Estonia",
"Ethiopia",
"Falkland Islands",
"Faroe Islands",
"Fiji",
"Finland",
"France",
"French Guiana",
"French Polynesia",
"Gabon",
"Gambia, The",
"Gaza Strip",
"Georgia",
"Germany",
"Ghana",
"Gibraltar",
"Greece",
"Greenland",
"Grenada",
```

```
"Guadeloupe",
"Guam",
"Guatemala",
"Guinea",
"Guinea-Bissau",
"Guyana",
"Haiti",
"Honduras",
"Hong Kong",
"Hungary",
"Iceland",
"India",
"Indonesia",
"Iran",
"Iraq",
"Ireland",
"Israel",
"Italy",
"Jamaica",
"Japan",
"Jordan",
"Kazakhstan",
"Korea, North",
"Korea, South",
"Kenya",
"Kiribati",
"Kuwait",
"Kyrgyzstan",
"Laos",
"Latvia",
"Lebanon",
"Lesotho",
"Liberia",
"Libya",
"Liechtenstein",
"Lithuania",
"Luxembourg",
"Macau",
"Macedonia, The Former Yugoslav Republic of",
"Madagascar",
"Malawi",
"Malaysia",
"Maldives",
"Mali",
"Malta",
"Marshall Islands",
"Martinique",
"Mauritania",
"Mexico",
"Micronesia",
"Moldova",
"Monaco",
"Mongolia",
"Montserrat",
"Morocco",
"Mozambique",
"Namibia",
"Nauru",
"Nepal",
"Netherlands",
```

```
"Netherlands Antilles",
"New Caledonia",
"New Zealand",
"Nicaragua",
"Niger",
"Nigeria",
"Niue",
"Northern Mariana Islands",
"Norway",
"Oman",
"Pakistan",
"Palau",
"Panama",
"Papua New Guinea",
"Paraguay",
"Peru",
"Philippines",
"Pitcairn Islands",
"Poland",
"Portugal",
"Puerto Rico",
"Qatar",
"Reunion",
"Romania",
"Russia",
"Rwanda",
"Saint Kitts and Nevis",
"Saint Lucia",
"Saint Pierre and Miquelon",
"Saint Vincent and the Grenadines",
"Samoa",
"San Marino",
"Sao Tome and Principe",
"Saudi Arabia",
"Senegal",
"Serbia and Montenegro",
"Seychelles",
"Sierra Leone",
"Singapore",
"Slovakia",
"Slovenia",
"Solomon Islands",
"Somalia",
"South Africa",
"Spain",
"Sri Lanka",
"Sudan",
"Suriname",
"Swaziland",
"Sweden",
"Switzerland",
"Syria",
"Taiwan",
"Tajikistan",
"Tanzania",
"Thailand",
"Togo",
"Tokelau",
"Tonga",
"Trinidad and Tobago",
```

```
    "Tunisia",
    "Turkey",
    "Turkmenistan",
    "Turks and Caicos Islands",
    "Tuvalu",
    "Uganda",
    "Ukraine",
    "United Arab Emirates",
    "United Kingdom",
    "Uruguay",
    "Uzbekistan",
    "Vanuatu",
    "Venezuela",
    "Vietnam",
    "Virgin Islands",
    "Wallis and Futuna",
    "West Bank",
    "Western Sahara",
    "Yemen",
    "Zambia",
    "Zimbabwe"
);

function print_country_select($selected_country="None") {
    global $countries;

    //See if we can find the user's country and select it as default:
    /*$gi = geoip_open("../inc/GeoIP.dat",GEOIP_STANDARD);
    $geoip_country =
geoip_country_name_by_addr($gi,$_SERVER["REMOTE_ADDR"]);
    geoip_close($gi);

    if ($selected_country=="") $selected_country="None";
    if ($selected_country=="None" and $geoip_country!=""){
        $selected_country=$geoip_country;
    }
    echo "selected: $selected_country\n";
*/
    $numCountries = count($countries);
    for ($i=0; $i<$numCountries; $i++) {
        $country = $countries[$i];
        $selected = ($selected_country == $country ? "selected":"");
        echo "<option value=\"$country\"
$selected>$country</option>\n";
    }
}

function is_valid_country($country) {
    global $countries;
    return in_array($country, $countries);
}

?>
```

D APÉNDICE D: CARACTERÍSTICAS Y COSTOS DEL SERVIDOR

Procesador	Intel(R) Pentium(R) 4 2.80GHz Socket 478 Bus 800 MHz
Mainboard	Intel D875 PBZLK P4 RAID 800 MHz, LAN Gigabit
Memoria RAM	1024 MB DDR PC 400 KINGSTON
Disco Duro	120 GB Serial ATA 7200 RPM
Video	NVIDIA GFORCE 5200 128 MB
CDROM	CDR 52 x
Periféricos	Unidad Floppy 3 1!2" , Mouse óptico, Teclado Multimedia
Monitor	Samsung 15"
PRECIO	**1355 USD + IVA**

E APÉNDICE E: DETALLE DE LAS CARACTERÍSTICAS DE LAS ESTACIONES CLIENTE

E.1 Computadora #1:

Nombre	TITAN
Sistema Operativo	Microsoft Windows XP Professional
Procesador	Intel(R) Pentium(R) 4 3.20GHz
Memoria RAM	1021.73 MB
Disco Duro	58.59 GB
Espacio En Disco	45.31 GB

BOINC BENCHMARK	
FLOPS	559992852
IOPS	715004572

CPU BENCHMARK (MODO EXCLUSIVO)	
MFLOPS	392.00
MIOPS	385.04

E.2 Computadora #2:

Nombre	REA
Sistema Operativo	Microsoft Windows XP Professional
Procesador	Intel(R) Pentium(R) 4 3.20GHz
Memoria RAM	512 MB
Disco Duro	19.53 GB
Espacio En Disco	5.12 GB

BOINC BENCHMARK	
FLOPS	335262674
IOPS	456600881

CPU BENCHMARK (MODO EXCLUSIVO)	
MFLOPS	454.54
MIOPS	207.62

E.3 Computadora #3:

Nombre	SAGITARIO
Sistema Operativo	Microsoft Windows XP Professional
Procesador	Intel(R) Pentium(R) 4 2.80GHz
Memoria RAM	446.48 MB
Disco Duro	34.18 GB
Espacio En Disco	25.45 GB

BOINC BENCHMARK	
FLOPS	994785980
IOPS	1589231262

CPU BENCHMARK (MODO EXCLUSIVO)	
MFLOPS	352.26
MIOPS	246.97

E.4 Computadora #4:

Nombre	LAPTOP
Sistema Operativo	Microsoft Windows XP Professional
Procesador	Intel(R) Pentium(R) M 1.50GHz
Memoria RAM	478.42 MB
Disco Duro	55.88 GB
Espacio En Disco	24.99 GB

BOINC BENCHMARK	
FLOPS	621886792
IOPS	725433962

CPU BENCHMARK (MODO EXCLUSIVO)	
MFLOPS	272.00
MIOPS	320.00

E.5 Computadora #5:

Nombre	PC ESCRITORIO
Sistema Operativo	Microsoft Windows XP Professional
Procesador	Intel(R) Pentium(R) 4 1.80GHz
Memoria RAM	604MB
Disco Duro	20 GB
Espacio En Disco	476 GB

BOINC BENCHMARK	
FLOPS	549235153
IOPS	444154947

CPU BENCHMARK (MODO EXCLUSIVO)	
MFLOPS	257.08
MIOPS	117.08

F APÉNDICE F: ARCHIVOS FUENTES DE LA APLICACIÓN DISTRIBUIDA

F.1 LargeNumber.h

```
//LargeNumber.h
#ifndef __LARGENUMBER_H__
#define __LARGENUMBER_H__

#include <vector>
#include <string>

using namespace std;

class CLargeNumber
{
public:
   //Default Constructor
   CLargeNumber() : m_cSign(1) { m_oNumber.push_back(0); }
   //Copy Constructor - Default OK
   //CLargeNumber(CLargeNumber     const&     rcoLargeNumber)     :
m_cSign(rcoLargeNumber.m_cSign),
m_oNumber(rcoLargeNumber.m_oNumber) {}

   //Constructor From a Number
   CLargeNumber(int iNumber);
   //Constructor From a String
   CLargeNumber(string const& rostrNumber);
   //Constructor From Members
   CLargeNumber(char   cSign,   vector<char>   const&   rcoNumber)   :
m_cSign(cSign), m_oNumber(rcoNumber) {}

   //Assignment Operator - Default OK
   //CLargeNumber& operator=(CLargeNumber const& roLN);

protected:
   //Auxiliary class Functions:
   //Build from unsigned long
   static void Build(unsigned uN, vector<char>& rvN);
   //Build from string
   static void Build(string const& rostrNumber, vector<char>& rvN);
   //Cleaning
   static void Clean(vector<char>& rvN);
   //Comparison Function
   static int Compare(vector<char> const& rcvN1, vector<char> const&
rcvN2);
   //Addition
   static  void  Add(vector<char>  const&  rcvN1,  vector<char>  const&
rcvN2, vector<char>& rvNRes);
   //Subtraction
   static  void  Subtract(vector<char>  const&  rcvN1,  vector<char>
const& rcvN2, vector<char>& rvNRes);
   //Product with one digit
```

```
  static void Multiply(vector<char> const& rcvN, char c,
vector<char>& rvNRes);
  //Shift Left
  static void ShiftLeft(vector<char>& rvN, int iLeft);
  //Multiplication
  static void Multiply(vector<char> const& rcvN1, vector<char>
const& rcvN2, vector<char>& rvNRes);
  //Get the Position of the most significant Digit
  static int Position(vector<char> const& rcvN);
  //Compute a Power of 10
  static void Pow10(unsigned uPow, vector<char>& rvNRes);
  //Division
  static void Divide(vector<char> const& rcvN1, vector<char> const&
rcvN2, vector<char>& rvQ, vector<char>& rvR);

public:
  //Transform to a string
  string ToString() const;
  //Operators
  //Equality Operator
  bool operator==(CLargeNumber const& roLN);
  //Inequality Operator
  bool operator!=(CLargeNumber const& roLN);
  CLargeNumber& operator-();
  bool operator<(CLargeNumber const& roLN) const;
  bool operator>(CLargeNumber const& roLN) const;
  bool operator<=(CLargeNumber const& roLN) const;
  bool operator>=(CLargeNumber const& roLN) const;
  CLargeNumber operator+(CLargeNumber const& roLN) const;
  CLargeNumber operator-(CLargeNumber const& roLN) const;
  CLargeNumber operator*(CLargeNumber const& roLN) const;
  CLargeNumber operator/(CLargeNumber const& roLN) const;
  CLargeNumber operator%(CLargeNumber const& roLN) const;
  CLargeNumber& operator+=(CLargeNumber const& roLN);
  CLargeNumber& operator-=(CLargeNumber const& roLN);
  CLargeNumber& operator*=(CLargeNumber const& roLN);
  CLargeNumber& operator/=(CLargeNumber const& roLN);
  CLargeNumber& operator%=(CLargeNumber const& roLN);
  //Convertion operator
  operator int() const;
  //Square Root
  CLargeNumber SquareRoot() const;

private:
  //-1 - Negative, +1 - Positive or zero
  char m_cSign;
  vector<char> m_oNumber;
};

#endif //__LARGENUMBER_H__
```

F.2 LargeNumber.cpp

```
#include "LargeNumber.h"
#include <stdexcept>

//=============================================================
=======
//Auxiliary Functions:
//Build from unsigned long
void CLargeNumber::Build(unsigned uN, vector<char>& rvN)
{
  rvN.clear();
  if(0 == uN)
         rvN.push_back(0);
  else
         for( ;uN>0; )
         {
                rvN.push_back(uN % 10);
                uN /= 10;
         }
}

//Build from string
void     CLargeNumber::Build(string     const&     rostrNumber,
vector<char>& rvN)
{
  rvN.clear();
  for(int i=rostrNumber.size()-1; i>=0; i--)
  {
         if(rostrNumber[i]<'0' || rostrNumber[i]>'9')
                break;
         else
                rvN.push_back(rostrNumber[i]-'0');
  }
  Clean(rvN);
}

//Cleaning
void CLargeNumber::Clean(vector<char>& rvN)
{
  //Eliminate all leading 0s
  vector<char>::iterator it = rvN.end();
  while(it != rvN.begin())
  {
         it--;
         if(*it != 0)
                break;
  }
  rvN.erase(it+1, rvN.end());
}

//Comparison Function
```

```
int     CLargeNumber::Compare(vector<char>     const&     rcvN1,
vector<char> const& rcvN2)
{
  if(rcvN1.size() == rcvN2.size())
  {
         for(int i=rcvN1.size()-1; i>=0; i--)
         {
                if(rcvN1[i] == rcvN2[i])
                       continue;
                return (rcvN1[i] < rcvN2[i]) ? -1 : 1;
         }
         return 0; //are equal
  }
  else
         return (rcvN1.size() < rcvN2.size()) ? -1 : 1;
}

//Addition
void  CLargeNumber::Add(vector<char>  const&  rcvN1,  vector<char>
const& rcvN2, vector<char>& rvNRes)
{
  rvNRes.clear();
  //Local copies
  vector<char> vN1 = rcvN1;
  vector<char> vN2 = rcvN2;
  int iSize1 = vN1.size();
  int iSize2 = vN2.size();
  int i, iSize;
  //Fill with '0'
  if(iSize1 > iSize2)
  {
         for(i=iSize2; i<iSize1; i++)
                vN2.push_back(0);
         iSize = iSize1;
  }
  else
  {
         for(i=iSize1; i<iSize2; i++)
                vN1.push_back(0);
         iSize = iSize2;
  }
  int iC=0, iR;
  for(i=0; i<iSize; i++)
  {
         iR = vN1[i] + vN2[i] + iC;
         if(iR > 9)
         {
                iR -= 10;
                iC = 1;
         }
         else
                iC = 0;
         rvNRes.push_back(iR);
  }
```

```
  if(iC > 0)
        rvNRes.push_back(iC);
}

//Subtraction
void    CLargeNumber::Subtract(vector<char>    const&    rcvN1,
vector<char> const& rcvN2, vector<char>& rvNRes)
{
  rvNRes.clear();
  //Local copy
  vector<char> vN1 = rcvN1;
  for(int i=0; i<rcvN2.size(); i++)
  {
        if(rcvN2[i] > vN1[i])
        {
              vN1[i] += 10;
              for(int j=i+1; j<vN1.size(); j++)
              {
                    if(vN1[j] > 0)
                    {
                          vN1[j]--;
                          break;
                    }
                    else
                          vN1[j] = 9;
              }
              vector<char>::iterator it = vN1.end();
              it--;
              if(0 == *it)
                    vN1.erase(it);
        }
        rvNRes.push_back(vN1[i]-rcvN2[i]);
  }
  for( ;i<vN1.size(); i++)
        rvNRes.push_back(vN1[i]);
  Clean(rvNRes);
}

//Product with one digit
void  CLargeNumber::Multiply(vector<char>  const&  rcvN,  char  c,
vector<char>& rvNRes)
{
  rvNRes.clear();
  char t, s;
  t = 0;
  if(0 == c)
        rvNRes.push_back(0);
  else
  {
        for(int i=0; i<rcvN.size(); i++)
        {
              (s = c * rcvN[i] + t) > 9 ? (t = s/10, s %= 10)
: (t = 0);
              rvNRes.push_back(s);
```

```
		}
		if(t>0)
			rvNRes.push_back(t);
	}
}

//Shift Left
void CLargeNumber::ShiftLeft(vector<char>& rvN, int iLeft)
{
	vector<char>::iterator i;
	for(int j=0; j<iLeft; j++)
	{
		i = rvN.begin();
		rvN.insert(i, 0);
	}
}

//Multiplication
void	CLargeNumber::Multiply(vector<char>	const&	rcvN1,
vector<char> const& rcvN2, vector<char>& rvNRes)
{
	vector<char> vZero;
	Build("0", vZero);
	if( 0 == Compare(rcvN1, vZero) || 0 == Compare(rcvN2, vZero))
	{
		rvNRes = vZero;
		return;
	}
	rvNRes.clear();
	vector<char> vDummy;
	for(int i=0; i<rcvN2.size(); i++)
		if(rcvN2[i] > 0)
		{
			Multiply(rcvN1, rcvN2[i], vDummy);
			ShiftLeft(vDummy, i);
			if(0 == i)
				rvNRes = vDummy;
			else
			{
				vector<char> vDummy1;
				Add(rvNRes, vDummy, vDummy1);
				rvNRes = vDummy1;
			}
		}
}

//Get the Position of the most significant Digit
int CLargeNumber::Position(vector<char> const& rcvN)
{
	int iPos = rcvN.size();
	if(0 == iPos)
		iPos = 1;
	return iPos;
}
```

```
//Compute a Power of 10
void CLargeNumber::Pow10(unsigned uPow, vector<char>& rvNRes)
{
  rvNRes.clear();
  rvNRes.push_back(1);
  vector<char>::iterator i;
  for(int j=0; j<uPow; j++)
  {
         i = rvNRes.begin();
         rvNRes.insert(i, 0);
  }
}

//Division
void     CLargeNumber::Divide(vector<char>     const&     rcvN1,
vector<char> const& rcvN2, vector<char>& rvQ, vector<char>& rvR)
{
  rvQ.clear();
  rvR.clear();
  if(Compare(rcvN1, rcvN2) >= 0)
  {
         vector<char> vDummy;
         vector<char> vTmp = rcvN2;
         int iPos = Position(rcvN1)-Position(rcvN2)-1;
         vector<char> vIncr;
         if(iPos > 0)
         {
                ShiftLeft(vTmp, iPos);
                Pow10(iPos, vIncr);
         }
         else
                Build(1, vIncr);
         rvR = rcvN1;
         Build(unsigned(0), rvQ);
         while(Compare(rvR, vTmp) >= 0)
         {
                Add(rvQ, vIncr, vDummy);
                rvQ = vDummy;
                Subtract(rvR, vTmp, vDummy);
                rvR = vDummy;
         }
         if(Compare(rvR, rcvN2) >= 0)
         {
                vector<char> vQ;
                vector<char> vR;
                Divide(rvR, rcvN2, vQ, vR);
                rvR = vR;
                Add(rvQ, vQ, vDummy);
                rvQ = vDummy;
         }
  }
  else
  {
```

```
        Build(unsigned(0), rvQ);
        rvR = rcvN1;
  }
}

//=============================================================
=======

//Constructor From a Number
CLargeNumber::CLargeNumber(int iNumber)
{
  if(iNumber < 0)
  {
        m_cSign = -1;
        iNumber = -iNumber;
  }
  else
        m_cSign = 1;
  Build(iNumber, m_oNumber);
}

//Constructor From a String
CLargeNumber::CLargeNumber(string const& rostrNumber)
{
  //Eliminating leading and trailing blanks
  int iEnd;
  iEnd = rostrNumber.size()-1;
  for(; (rostrNumber[iEnd]==' ')&&(iEnd>=0); iEnd--)
        ;
  if(iEnd < 0)
  {
        m_cSign = 1;
        Build(0, m_oNumber);
        return;
  }
  int iBeg;
  for(iBeg=0; ' '==rostrNumber[iBeg]; iBeg++)
        ;
  string ostrNumber = rostrNumber.substr(iBeg, iEnd-iBeg+1);
  iBeg = 0;
  if('-' == ostrNumber[0])
  {
        m_cSign = -1;
        iBeg = 1;
  }
  else
        m_cSign = 1;
  Build(ostrNumber.c_str()+iBeg, m_oNumber);
}

//Transform to a string
string CLargeNumber::ToString() const
{
  if(0 == m_oNumber.size())
```

```
        return "0";
  string ostr;
  if(-1 == m_cSign)
        ostr += '-';
  vector<char>::const_reverse_iterator          rIter          =
m_oNumber.rbegin();
  for(; rIter != m_oNumber.rend(); rIter++)
        ostr += *rIter+'0';
  return ostr;
}

//Equality Operator
bool CLargeNumber::operator==(CLargeNumber const& roLN)
{
  if(m_oNumber==roLN.m_oNumber && m_cSign==roLN.m_cSign)
        return true;
  else
        return false;
}

//Inequality Operator
bool CLargeNumber::operator!=(CLargeNumber const& roLN)
{
  return !(operator==(roLN));
}

CLargeNumber& CLargeNumber::operator-()
{
  m_cSign = -m_cSign;
  return *this;
}

bool CLargeNumber::operator<(CLargeNumber const& roLN) const
{
  if(-1==m_cSign && 1==roLN.m_cSign)
        return true;
  else if(1==m_cSign && -1==roLN.m_cSign)
        return false;
  else if(-1==m_cSign && -1==roLN.m_cSign)
        return (-1==Compare(roLN.m_oNumber, m_oNumber));
  else //if(1==m_cSign && 1==roLN.m_cSign)
        return (-1==Compare(m_oNumber, roLN.m_oNumber));
}

bool CLargeNumber::operator>(CLargeNumber const& roLN) const
{
  if(-1==m_cSign && 1==roLN.m_cSign)
        return false;
  else if(1==m_cSign && -1==roLN.m_cSign)
        return true;
  else if(-1==m_cSign && -1==roLN.m_cSign)
        return (-1==Compare(m_oNumber, roLN.m_oNumber));
  else //if(1==m_cSign && 1==roLN.m_cSign)
        return (-1==Compare(roLN.m_oNumber, m_oNumber));
```

```
}

bool CLargeNumber::operator<=(CLargeNumber const& roLN) const
{
  return !operator>(roLN);
}

bool CLargeNumber::operator>=(CLargeNumber const& roLN) const
{
  return !operator<(roLN);
}

CLargeNumber  CLargeNumber::operator+(CLargeNumber  const&  roLN)
const
{
  CLargeNumber oLNRes;
  if(1 == m_cSign && 1 == roLN.m_cSign)
  {
         oLNRes.m_cSign = 1;
         Add(m_oNumber, roLN.m_oNumber, oLNRes.m_oNumber);
  }
  else if(-1 == m_cSign && -1 == roLN.m_cSign)
  {
         oLNRes.m_cSign = -1;
         Add(m_oNumber, roLN.m_oNumber, oLNRes.m_oNumber);
  }
  else
  {
         int iComp = Compare(m_oNumber, roLN.m_oNumber);
         if(0 == iComp)
                return CLargeNumber(0L);
         else if(-1 == iComp)
                Subtract(roLN.m_oNumber,              m_oNumber,
oLNRes.m_oNumber);
         else
                Subtract(m_oNumber,              roLN.m_oNumber,
oLNRes.m_oNumber);
         if(1 == m_cSign && -1 == roLN.m_cSign)
         {
                if(-1 == iComp)
                       oLNRes.m_cSign = -1;
                else
                       oLNRes.m_cSign = 1;
         }
         else
         {
                if(-1 == iComp)
                       oLNRes.m_cSign = 1;
                else
                       oLNRes.m_cSign = -1;
         }
  }
  return oLNRes;
}
```

```
CLargeNumber  CLargeNumber::operator-(CLargeNumber  const&  roLN)
const
{
  CLargeNumber oLNRes;
  if(1 == m_cSign && -1 == roLN.m_cSign)
  {
         oLNRes.m_cSign = 1;
         Add(m_oNumber, roLN.m_oNumber, oLNRes.m_oNumber);
  }
  else if(-1 == m_cSign && 1 == roLN.m_cSign)
  {
         oLNRes.m_cSign = -1;
         Add(m_oNumber, roLN.m_oNumber, oLNRes.m_oNumber);
  }
  else
  {
         int iComp = Compare(m_oNumber, roLN.m_oNumber);
         if(0 == iComp)
         {
                return CLargeNumber(0);
         }
         else if(-1 == iComp)
                Subtract(roLN.m_oNumber,              m_oNumber,
oLNRes.m_oNumber);
         else
                Subtract(m_oNumber,              roLN.m_oNumber,
oLNRes.m_oNumber);
         if(1 == m_cSign && 1 == roLN.m_cSign)
         {
                if(-1 == iComp)
                       oLNRes.m_cSign = -1;
                else
                       oLNRes.m_cSign = 1;
         }
         else
         {
                if(-1 == iComp)
                       oLNRes.m_cSign = 1;
                else
                       oLNRes.m_cSign = -1;
         }
  }
  return oLNRes;
}

CLargeNumber  CLargeNumber::operator*(CLargeNumber  const&  roLN)
const
{
  CLargeNumber oLNRes;
  oLNRes.m_cSign = m_cSign * roLN.m_cSign;
  Multiply(m_oNumber, roLN.m_oNumber, oLNRes.m_oNumber);
  return oLNRes;
}
```

```
CLargeNumber  CLargeNumber::operator/(CLargeNumber  const&  roLN)
const
{
  if(roLN < CLargeNumber("0"))
         //Throw an exception
         throw overflow_error("ERROR: Overflow in operator/!");
  if(0==Compare(m_oNumber, CLargeNumber("0").m_oNumber))
         return CLargeNumber("0");
  CLargeNumber oLNQ, oLNDummy;
  Divide(m_oNumber,        roLN.m_oNumber,        oLNQ.m_oNumber,
oLNDummy.m_oNumber);
  oLNQ.m_cSign                =                (-1==m_cSign)&&(-
1==roLN.m_cSign)||(1==m_cSign)&&(1==roLN.m_cSign) ? 1 : -1;
  return oLNQ;
}

CLargeNumber  CLargeNumber::operator%(CLargeNumber  const&  roLN)
const
{
  if(roLN < CLargeNumber("0"))
         //Throw an exception
         throw overflow_error("ERROR: Overflow in operator%!");
  if(0==Compare(m_oNumber, CLargeNumber("0").m_oNumber))
         return CLargeNumber("0");
  CLargeNumber oLNDummy, oLNR;
  Divide(m_oNumber,     roLN.m_oNumber,     oLNDummy.m_oNumber,
oLNR.m_oNumber);
  oLNR.m_cSign = m_cSign;
  return oLNR;
}

CLargeNumber& CLargeNumber::operator+=(CLargeNumber const& roLN)
{
  CLargeNumber oLNRes;
  if(1 == m_cSign && 1 == roLN.m_cSign)
  {
         m_cSign = 1;
         Add(m_oNumber, roLN.m_oNumber, oLNRes.m_oNumber);
  }
  else if(-1 == m_cSign && -1 == roLN.m_cSign)
  {
         m_cSign = -1;
         Add(m_oNumber, roLN.m_oNumber, oLNRes.m_oNumber);
  }
  else
  {
         int iComp = Compare(m_oNumber, roLN.m_oNumber);
         if(0 == iComp)
         {
                *this = CLargeNumber(0);
                return *this;
         }
         else if(-1 == iComp)
```

```
                Subtract(roLN.m_oNumber,              m_oNumber,
oLNRes.m_oNumber);
        else
                Subtract(m_oNumber,              roLN.m_oNumber,
oLNRes.m_oNumber);
        if(1 == m_cSign && -1 == roLN.m_cSign)
        {
                if(-1 == iComp)
                        m_cSign = -1;
                else
                        m_cSign = 1;
        }
        else
        {
                if(-1 == iComp)
                        m_cSign = 1;
                else
                        m_cSign = -1;
        }
  }
  m_oNumber = oLNRes.m_oNumber;
  return *this;
}

CLargeNumber& CLargeNumber::operator-=(CLargeNumber const& roLN)
{
  CLargeNumber oLNRes;
  if(1 == m_cSign && -1 == roLN.m_cSign)
  {
        m_cSign = 1;
        Add(m_oNumber, roLN.m_oNumber, oLNRes.m_oNumber);
  }
  else if(-1 == m_cSign && 1 == roLN.m_cSign)
  {
        m_cSign = -1;
        Add(m_oNumber, roLN.m_oNumber, oLNRes.m_oNumber);
  }
  else
  {
        int iComp = Compare(m_oNumber, roLN.m_oNumber);
        if(0 == iComp)
        {
                *this = CLargeNumber(0);
                return *this;
        }
        else if(-1 == iComp)
                Subtract(roLN.m_oNumber,              m_oNumber,
oLNRes.m_oNumber);
        else
                Subtract(m_oNumber,              roLN.m_oNumber,
oLNRes.m_oNumber);
        if(1 == m_cSign && 1 == roLN.m_cSign)
        {
                if(-1 == iComp)
```

```
                        m_cSign = -1;
                else
                        m_cSign = 1;
        }
        else
        {
                if(-1 == iComp)
                        m_cSign = 1;
                else
                        m_cSign = -1;
        }
  }
  m_oNumber = oLNRes.m_oNumber;
  return *this;
}

CLargeNumber& CLargeNumber::operator*=(CLargeNumber const& roLN)
{
  CLargeNumber oLNRes;
  m_cSign = m_cSign * roLN.m_cSign;
  Multiply(m_oNumber, roLN.m_oNumber, oLNRes.m_oNumber);
  *this = oLNRes;
  return *this;
}

CLargeNumber& CLargeNumber::operator/=(CLargeNumber const& roLN)
{
  if(roLN<CLargeNumber("0"))
        //Throw an exception
        throw  overflow_error("ERROR:  Overflow  in  operator
/=!");
  if(0==Compare(m_oNumber, CLargeNumber("0").m_oNumber))
  {
        *this = CLargeNumber("0");
  }
  else
  {
        CLargeNumber oLNQ, oLNDummy;
        Divide(m_oNumber,    roLN.m_oNumber,    oLNQ.m_oNumber,
oLNDummy.m_oNumber);
        oLNQ.m_cSign            =            (-1==m_cSign)&&(-
1==roLN.m_cSign)||(1==m_cSign)&&(1==roLN.m_cSign) ? 1 : -1;
        *this = oLNQ;
  }
  return *this;
}

CLargeNumber& CLargeNumber::operator%=(CLargeNumber const& roLN)
{
  if(roLN<CLargeNumber("0"))
        //Throw an exception
        throw      overflow_error("ERROR:      Overflow      in
operator%=!");
  if(0==Compare(m_oNumber, CLargeNumber("0").m_oNumber))
```

```
  {
        *this = CLargeNumber("0");
  }
  else
  {
        CLargeNumber oLNDummy, oLNR;
        Divide(m_oNumber,  roLN.m_oNumber,  oLNDummy.m_oNumber,
oLNR.m_oNumber);
        oLNR.m_cSign = m_cSign;
        *this = oLNR;
  }
  return *this;
}

//Convertion operator
CLargeNumber::operator int() const
{
  CLargeNumber oLNMin(INT_MIN);
  CLargeNumber oLNMax(INT_MAX);
  if(operator<(oLNMin) || operator>(oLNMax))
        //Throw an exception
        throw   domain_error("ERROR:   Domain   in   operator
int()!");
  unsigned uPow10 = 1;
  int iRes = 0;
  if(-1==m_cSign)
  {
        for(int i=0; i<m_oNumber.size(); i++)
        {
              iRes -= m_oNumber[i]*uPow10;
              uPow10 *= 10;
        }
  }
  else
  {
        for(int i=0; i<m_oNumber.size(); i++)
        {
              iRes += m_oNumber[i]*uPow10;
              uPow10 *= 10;
        }
  }
  return iRes;
}

//Square Root
CLargeNumber CLargeNumber::SquareRoot() const
{
  if(operator<(CLargeNumber("0")))
        //Throw an exception
        throw  invalid_argument("ERROR:  Negative  Number  in
function SquareRoot()!");
  //Initialize
  CLargeNumber oNumber1 = *this;
  CLargeNumber oLN0(0);
```

```
  CLargeNumber oLN2(2);
  CLargeNumber oLN10(10);
  CLargeNumber oLN100(100);
  CLargeNumber oSqR(oLN10);
  while((oNumber1 /= oLN100) > oLN0)
    oSqR *= oLN10;
  //Recursive Algorithm
  CLargeNumber oSqROld;
  while(true)
  {
         oSqROld = oSqR;
         oSqR = (oSqR + (*this)/oSqR)/oLN2;
         if(oSqR >= oSqROld)
         {
                oSqR = oSqROld;
                return oSqR;
         }
  }
}

//=============================================================
=======
```

F.3 windows_opengl.c

```
//Manejo de la parte gráfica de la aplicacion de desencriptacion
//de claves RSA

#include "stdafx.h"

#include "boinc_api.h"
#include "graphics_api.h"
#include "app_ipc.h"
#include "util.h"
#include "win_util.h"

//Librerias para el dibujo del logo de la ESPOL
#include "../apps/rgbpixmap.h"
#include <math.h>
#define PI 3.1415926535897
RGBpixmap pic;

//Posiciones relativas
double POS_PILARZ = -5.5;

//Utilidades para graficos de BOINC
#include "gutil.h";
#include "graphics_data.h"

//#include "win_idle_tracker.h"
```

```
// application needs to define mouse handlers
//

//Estructura que guarda el contenido de una unidad de trabajo
struct CONTENIDO_WU
{
  char n[110]; //Numero n, que forma parte de la clave publica
  char e[110]; //Numero e que forma parte de la clave publica
  char fact_inicial[110]; //Desde donde se comienza a buscar
  char fact_final[110]; //hasta donde se busca
};
typedef struct CONTENIDO_WU WorkUnit;

//Variables compartidas
extern APP_INIT_DATA aid;
extern double porcentaje;
//extern WorkUnit infoWu;
extern string temporal;
extern string numero_actual;
extern WorkUnit infoWu;

extern void boinc_app_mouse_button(int x, int y, int which, bool
is_down);
extern void boinc_app_mouse_move(int x, int y, bool left, bool
middle, bool right);

#define BOINC_WINDOW_CLASS_NAME "BOINC_app"

static HDC                    hDC=NULL;
static HGLRC           hRC=NULL;
static HWND                   hWnd=NULL;           // Holds Our
Window Handle
static HINSTANCE hInstance;           //  Holds  The  Instance  Of
The Application
static RECT                   rect = {50, 50, 50+640, 50+480};
static int                    current_graphics_mode          =
MODE_HIDE_GRAPHICS;
static POINT           mousePos;
static UINT                   m_uEndSSMsg;
static HDC myhDC;
BOOL            win_loop_done;

static bool visible = true;

//Objetos graficos
PROGRESS myProgress; //Barra de progreso

//Variables para los graficos
double max_fps;                // max frames per second
double max_cpu;                // max pct of CPU
double graph_alpha;
double start_hue, hue_change;   // determine graph colors
double viewpoint_distance = 1.0;
extern int width, height;
```

```
int angle=0;
int dir=0;
void KillWindow();

void SetupPixelFormat(HDC hDC) {
   int nPixelFormat;

   static PIXELFORMATDESCRIPTOR pfd = {
         sizeof(PIXELFORMATDESCRIPTOR),   // size of structure.
         1,                               // always 1.
         PFD_DRAW_TO_WINDOW |             // support window
         PFD_SUPPORT_OPENGL |             // support OpenGl
         PFD_DOUBLEBUFFER,                     // support double
buffering
         PFD_TYPE_RGBA,                   // support RGBA
         32,                              // 32 bit color mode
         0, 0, 0, 0, 0, 0,                // ignore color bits
         0,                               // no alpha buffer
         0,                               // ignore shift bit
         0,                                   // no accumulation
buffer
         0, 0, 0, 0,                      // ignore accumulation
bits.
         16,                                  // number of depth
buffer bits.
         0,                                 // number of stencil
buffer bits.
         0,                                       // 0 means no
auxiliary buffer
         PFD_MAIN_PLANE,                   // The main drawing
plane
         0,                               // this is reserved
         0,  0,  0 };                            // layer masks
ignored.



   // this chooses the best pixel format and returns index.
   nPixelFormat = ChoosePixelFormat(hDC, &pfd);

   // This set pixel format to device context.
   SetPixelFormat(hDC, nPixelFormat, &pfd);

   // Remember that its not important to fully understand the
pixel format,
   // just remember to include in all of your applications and
you'll be
   // good to go.
}

static void make_new_window(int mode) {
  RECT WindowRect = {0,0,0,0};
  //int width, height;
```

```
  DWORD dwExStyle;
  DWORD dwStyle;

  if (current_graphics_mode == MODE_FULLSCREEN) {
         HDC screenDC=GetDC(NULL);
         WindowRect.left = WindowRect.top = 0;
         WindowRect.right=GetDeviceCaps(screenDC, HORZRES);
         WindowRect.bottom=GetDeviceCaps(screenDC, VERTRES);
         ReleaseDC(NULL, screenDC);
         dwExStyle=WS_EX_TOPMOST;
         dwStyle=WS_POPUP;
         while(ShowCursor(false) >= 0);
  } else {
         WindowRect = rect;
         dwExStyle=WS_EX_APPWINDOW|WS_EX_WINDOWEDGE;
         dwStyle=WS_OVERLAPPEDWINDOW;
         while(ShowCursor(true) < 0);
  }

    if (!strlen(aid.app_name)) strcpy(aid.app_name, "Aplicacion
de BOINC");
  hWnd   =   CreateWindowEx(dwExStyle,   BOINC_WINDOW_CLASS_NAME,
aid.app_name,
         dwStyle|WS_CLIPSIBLINGS|WS_CLIPCHILDREN,
WindowRect.left, WindowRect.top,
         WindowRect.right-WindowRect.left,WindowRect.bottom-
WindowRect.top,
         NULL, NULL, hInstance, NULL
    );

  SetForegroundWindow(hWnd);

  GetCursorPos(&mousePos);

  hDC = GetDC(hWnd);
  myhDC=hDC;
  SetupPixelFormat(myhDC);

  if(!(hRC = wglCreateContext(hDC))) {
         ReleaseDC(hWnd, hDC);
         return;
  }

  if(!wglMakeCurrent(hDC, hRC)) {
         ReleaseDC(hWnd, hDC);
         wglDeleteContext(hRC);
         return;
  }

  width = WindowRect.right-WindowRect.left;
  height = WindowRect.bottom-WindowRect.top;

  if(current_graphics_mode      ==      MODE_FULLSCREEN      ||
current_graphics_mode == MODE_WINDOW) {
```

```
          ShowWindow(hWnd, SW_SHOW);
          SetFocus(hWnd);
  } else {
          KillWindow();
  }

    app_graphics_init();
}

void KillWindow() {
  wglMakeCurrent(NULL,NULL);  // release GL rendering context
  if (hRC) {
          wglDeleteContext(hRC);
          hRC=NULL;
  }

    if (hWnd && hDC) {
        ReleaseDC(hWnd,hDC);
  }
    hDC = NULL;

    if (hWnd) {
        DestroyWindow(hWnd);
  }
    hWnd = NULL;
}

// switch to the given graphics mode.  This is called:
// - on initialization
// - when get mode change msg (via shared mem)
// - when in SS mode and get user input
//
void SetMode(int mode) {

  if      (current_graphics_mode      !=      MODE_FULLSCREEN)
GetWindowRect(hWnd, &rect);

  KillWindow();

  current_graphics_mode = mode;

    if (mode != MODE_HIDE_GRAPHICS) {
        make_new_window(mode);
    }

    // tell the core client that we're entering new mode
    //
    if (app_client_shm) {
        app_client_shm->send_graphics_msg(
            APP_CORE_GFX_SEG,             GRAPHICS_MSG_SET_MODE,
current_graphics_mode
        );
    }
}
```

```
void parse_mouse_event(UINT uMsg, int& which, bool& down) {
  switch(uMsg) {
  case WM_LBUTTONDOWN: which = 0; down = true; break;
  case WM_MBUTTONDOWN: which = 1; down = true; break;
  case WM_RBUTTONDOWN: which = 2; down = true; break;
  case WM_LBUTTONUP: which = 0; down = false; break;
  case WM_MBUTTONUP: which = 1; down = false; break;
  case WM_RBUTTONUP: which = 2; down = false; break;
    }
}


LRESULT CALLBACK WndProc(
    HWND  hWnd,                 // Handle For This Window
    UINT  uMsg,                 // Message For This Window
    WPARAM      wParam,               //    Additional    Message
Information
    LPARAM      lParam                //    Additional    Message
Information
) {
  switch(uMsg)
  {
    //Mensajes que se manejan por parte del usuario
  //chalo6870
  case WM_ERASEBKGND:
         return 0;
  case WM_KEYDOWN:
  case WM_KEYUP:
  case WM_LBUTTONDOWN:
  case WM_MBUTTONDOWN:
  case WM_RBUTTONDOWN:
  case WM_LBUTTONUP:
  case WM_MBUTTONUP:
  case WM_RBUTTONUP:
  case WM_MOUSEMOVE:
        //Si actualmente esta en pantalla completa
         POINT cPos;
         GetCursorPos(&cPos);
         if(current_graphics_mode == MODE_FULLSCREEN)
         {
                if(cPos.x    !=    mousePos.x    ||    cPos.y    !=
mousePos.y) {
                       SetMode(MODE_HIDE_GRAPHICS);
                       PostMessage(HWND_BROADCAST,  m_uEndSSMsg,
0, 0);
                       //KillWindow();
                }
        }
        return 0;
  case WM_CLOSE:
        if (boinc_is_standalone()) {
            exit(0);
        } else {
```

```
            KillWindow();
            return 0;
        }
  case WM_PAINT:
         PAINTSTRUCT ps;
         RECT winRect;
         HDC pdc;
         pdc = BeginPaint(hWnd, &ps);
         GetClientRect(hWnd, &winRect);
         FillRect(pdc,                                    &winRect,
(HBRUSH)GetStockObject(BLACK_BRUSH));
         EndPaint(hWnd, &ps);
         return 0;
  case WM_SIZE:
        if ( SIZE_MINIMIZED == wParam ) {
            visible = FALSE;
        } else {
            visible = TRUE;
        }
         ReSizeGLScene(LOWORD(lParam),HIWORD(lParam));
         return 0;
  }
  return DefWindowProc(hWnd,uMsg,wParam,lParam);



BOOL reg_win_class() {
  WNDCLASS      wc;                                        //
Windows Class Structure

  hInstance                   = GetModuleHandle(NULL);
               // Grab An Instance For Our Window
  wc.style                    =   CS_HREDRAW   |   CS_VREDRAW   |
CS_OWNDC; // Redraw On Size, And Own DC For Window.
  wc.lpfnWndProc        = (WNDPROC) WndProc;
         // WndProc Handles Messages
  wc.cbClsExtra         = 0;
                        // No Extra Window Data
  wc.cbWndExtra         = 0;
                        // No Extra Window Data
  wc.hInstance          = hInstance;
               // Set The Instance
  wc.hIcon                    =   LoadIcon(NULL,   IDI_WINLOGO);
               // Load The Default Icon
  wc.hCursor                  =   LoadCursor(NULL,   IDC_ARROW);
               // Load The Arrow Pointer
  wc.hbrBackground      = NULL;
                        // No Background Required For GL
  wc.lpszMenuName               = NULL;
                        // We Don't Want A Menu
  wc.lpszClassName      = BOINC_WINDOW_CLASS_NAME;
         // Set The Class Name
```

```
  if (!RegisterClass(&wc))
                        // Attempt To Register The Window Class
  {
         MessageBox(NULL,"Failed   To   Register   The   Window
Class.","ERROR",MB_OK|MB_ICONEXCLAMATION);
         return FALSE;
                                // Return FALSE
  }

  return TRUE;
}

BOOL unreg_win_class() {
  if (!UnregisterClass(BOINC_WINDOW_CLASS_NAME,hInstance)) {
         MessageBox(NULL,"Could          Not          Unregister
Class.","SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
         hInstance=NULL;
                        // Set hInstance To NULL
  }

  return TRUE;
}

static VOID CALLBACK timer_handler(HWND, UINT, UINT, DWORD) {
  RECT rt;
  int width, height, new_mode, msg;
    if (app_client_shm) {
        if   (app_client_shm->get_graphics_msg(CORE_APP_GFX_SEG,
msg, new_mode)) {
            switch (msg) {
            case GRAPHICS_MSG_SET_MODE:
                SetMode(new_mode);
                break;
            case GRAPHICS_MSG_REREAD_PREFS:
                        // only reread graphics prefs if we have
a window open
                        //
                        switch(current_graphics_mode) {
                        case MODE_WINDOW:
                        case MODE_FULLSCREEN:
                               app_graphics_reread_prefs();
                               break;
                        }
                break;
            }
       }
  }
    if (!visible) return;
  if (current_graphics_mode == MODE_HIDE_GRAPHICS) return;
    if (!hWnd) return;

    // TODO: remove width, height from API
    //
  GetClientRect(hWnd, &rt);
```

```
  width = rt.right-rt.left;
  height = rt.bottom-rt.top;

    if (throttled_app_render(width, height, dtime())) {
        SwapBuffers(hDC);
    }
}

DWORD WINAPI win_graphics_event_loop( LPVOID gi ) {
  MSG                                 msg;          //   Windows
Message Structure
  m_uEndSSMsg = RegisterWindowMessage(STOP_SS_MSG);

  // Register window class and graphics mode message
  reg_win_class();

  SetTimer(NULL, 1, 100, &timer_handler);

    if (boinc_is_standalone()) {
        SetMode(MODE_WINDOW);
    } else {
        SetMode(MODE_HIDE_GRAPHICS);
    }
  win_loop_done = false;
  while(!win_loop_done) {
        if (GetMessage(&msg,NULL,0,0)) {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        } else {
            win_loop_done = true;
        }
  }

  unreg_win_class();

  SetEvent(hQuitEvent);       // Signal to the worker thread
that we're quitting
  return (DWORD)msg.wParam;          // Exit The thread
}

BOOL VerifyPassword(HWND hwnd)
{ // Under NT, we return TRUE immediately. This lets the saver
quit,
  // and the system manages passwords. Under '95, we call
VerifyScreenSavePwd.
  // This checks the appropriate registry key and, if necessary,
  // pops up a verify dialog
  OSVERSIONINFO    osv;    osv.dwOSVersionInfoSize=sizeof(osv);
GetVersionEx(&osv);
  if (osv.dwPlatformId==VER_PLATFORM_WIN32_NT) return TRUE;
  HINSTANCE hpwdcpl=::LoadLibrary("PASSWORD.CPL");
  if (hpwdcpl==NULL) {return TRUE;}
  typedef BOOL (WINAPI *VERIFYSCREENSAVEPWD)(HWND hwnd);
  VERIFYSCREENSAVEPWD VerifyScreenSavePwd;
```

```
  VerifyScreenSavePwd=

(VERIFYSCREENSAVEPWD)GetProcAddress(hpwdcpl,"VerifyScreenSavePwd
");
  if (VerifyScreenSavePwd==NULL)
  {
    FreeLibrary(hpwdcpl);return TRUE;
  }
  BOOL bres=VerifyScreenSavePwd(hwnd); FreeLibrary(hpwdcpl);
  return bres;
}


float txt_widths[256];

unsigned int MyCreateFont(char *fontName, int Size, int weight)
{
    // windows font
    HFONT hFont;
    unsigned int mylistbase =0;

    // Create space for 96 characters.
    mylistbase= glGenLists(256);

    if(stricmp(fontName, "symbol")==0) {
        hFont = CreateFont(
            Size, 0, 0, 0, FW_BOLD, FALSE, FALSE, FALSE,
            SYMBOL_CHARSET, OUT_TT_PRECIS, CLIP_DEFAULT_PRECIS,
            ANTIALIASED_QUALITY,  FF_DONTCARE  |  DEFAULT_PITCH,
fontName
        );
    } else {
        hFont = CreateFont(
            Size, 0, 0, 0, FW_BOLD, FALSE, FALSE, FALSE,
            ANSI_CHARSET, OUT_TT_PRECIS, CLIP_DEFAULT_PRECIS,
            ANTIALIASED_QUALITY,  FF_DONTCARE  |  DEFAULT_PITCH,
fontName
        );
    }

    if(!hFont) return -1;
    SelectObject(myhDC, hFont);
#if 1 //no idea why this has to be twice
    wglUseFontBitmaps(myhDC, 0, 256, mylistbase);
    wglUseFontBitmaps(myhDC, 0, 256, mylistbase);
#endif
#if 0

wglUseFontOutlines(hDC,0,255,mylistbase,0.0f,0.2f,WGL_FONT_POLYG
ONS,gmf);
#endif

     TEXTMETRIC met;
     GetTextMetrics(myhDC,&met);
```

```
GetCharWidthFloat(myhDC,met.tmFirstChar,met.tmLastChar,txt_width
s);

    return mylistbase;
}

float get_char_width(unsigned char c) {
  return txt_widths[c];
}


void createDisc() {
  GLint n=72;
  GLfloat radius = 1.5f;

  glNewList(2, GL_COMPILE);
         glEnable(GL_TEXTURE_2D);
         glBindTexture(GL_TEXTURE_2D, 2001);

         glBegin(GL_POLYGON);
                for (int i=0; i < n; i++) {
                       glNormal3f(0.0, 0.0, 1.0);
                       glTexCoord2f(cos((i+1)*2*PI/n)/2+0.5,
sin((i+1)*2*PI/n)/2+0.5);
                       glVertex3f(radius*cos((i+1)*2*PI/n),
radius*sin((i+1)*2*PI/n), 0.0);
                }
         glEnd();
         glDisable(GL_TEXTURE_2D);
  glEndList();
}

void createMesh() {
  GLint n=72;
  GLfloat radius = 1.5f;

  glPushMatrix();
  glTranslatef(0.0f, -0.1f, .25f);
  //mode_texture();
         glPushMatrix();
                //glRotated(180,0,0,1);
                glCallList(2);
                glTranslatef(0.0f, 0.0f, -.5f);
                glCallList(2);
         glPopMatrix();

         glPushMatrix();
                glEnable(GL_TEXTURE_2D);
                glBindTexture(GL_TEXTURE_2D, 2001);

                glBegin(GL_QUAD_STRIP);
                       for (int i=0; i <= n; i++) {
```

```
  glVertex3f(radius*cos((i+1)*2*PI/n),
radius*sin((i+1)*2*PI/n), 0.0);

  glVertex3f(radius*cos((i+1)*2*PI/n),
radius*sin((i+1)*2*PI/n), -.5);
                        }
                glEnd();
                glDisable(GL_TEXTURE_2D);
         glPopMatrix();
  glPopMatrix();
}



void setPreferences() {
  max_fps = 30;
    max_cpu = 50;
    graph_alpha = 0.8;
    start_hue = 0.2;
    hue_change = 0.5;
}


//Incilializa las luces
static void initlights() {
   GLfloat ambient[] = {1., 1., 1., 1.0};
   GLfloat position[] = {-13.0, 6.0, 20.0, 1.0};
   GLfloat dir[] = {-1, -.5, -3, 1.0};
   glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
   glLightfv(GL_LIGHT0, GL_POSITION, position);
   glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
}


//Dibuja los pilares estaticos para mostrar la info de la
aplicacion
void draw_pillars()
{
  float pos[3];
  GLfloat violet[] = {0.6, 0.3, .8, 1.0};
  GLfloat white[] = {1.0, 1.0, 1.0, 1.0};
  COLOR color;
    double lum = .5;
    double sat = .5;
    double hue = start_hue + hue_change/2.;
    if (hue > 1) hue -= 1;
    if (hue < 1) hue += 1;
    HLStoRGB(hue, lum, sat, color);
    color.a = 1.0;
  GLfloat graphColor[4] = {color.r,color.g,color.b,color.a};

  //Icicializamos las posiciones
  pos[0] = pos[1] = pos[2] = 0;
```

```
  //Pilar inferior
  mode_shaded(graphColor);
  pos[0] = -3.50; pos[1] = POS_PILARZ;
  drawCylinder(false, pos, 7.00, .10);
  drawSphere(pos, .10);
  pos[0] = 3.5; drawSphere(pos, .1);

  //Pilar Superior
  pos[0] = -3.50; pos[1] = 2.50 + POS_PILARZ;
  drawCylinder(false, pos, 7.00, .10);
  drawSphere(pos, .10);
  pos[0] = 3.5; drawSphere(pos, .1);

    //Pilar central horizontal
  pos[0] = 1.00; pos[1] =  1.25 + POS_PILARZ;
  drawCylinder(false, pos, 2.50, .10);
  pos[0] = 3.5; drawSphere(pos, .1);

  //Pilar central vertical
  pos[0] = 1.00; pos[1] =  POS_PILARZ;
  drawCylinder(true, pos, 2.50, .10);

    //Colocamos el texto cabecera de cada pilar
  mode_lines();
  glColor3f(1.,0.8, 0.);
  pos[0] = -3.5; pos[1] = 2.15 + POS_PILARZ;
  draw_text_line(pos,    0.18,    0.06,    "Datos    de    la
Desencriptacion");

  pos[0] = 1.2; pos[1] = 2.15 + POS_PILARZ;
  draw_text_line(pos, 0.18, 0.06, "Datos del Equipo");

  pos[0] = 1.2; pos[1] = 0.90 + POS_PILARZ;
  draw_text_line(pos, 0.18, 0.06, "Autores");

    //Nombre de los autores
  glColor3f(1.,1.,1.);
    pos[0] = 1.2; pos[1] = 0.65 + POS_PILARZ;
  draw_text_line(pos, 0.18, 0.06, "Gonzalo Luzardo");
  pos[0] = 1.2; pos[1] = 0.38 + POS_PILARZ;
  draw_text_line(pos, 0.18, 0.06, "Luis Vargas");

}



//Incilializa los graficos dinamicos
void init_dinamic_graphics()
{
  float pos[3];
  float outer[] = {1.0, 0., 0.2, 0.4};
    float inner[] = {0.2, 0.0, 0.7, 0.9};
```

```
  pos[0] = pos[1] = pos[2] = 0;

  //Inicializo la barra de progreso
  pos[0] = -3.5; pos[1] = 0.40 + POS_PILARZ;
  myProgress.init(pos, 4., 0.1, 0.08, outer,inner);

  //Incializo el disco con el logo de la ESPOL
  if(pic.readBMPFile("logoespol.bmp"))
  {
         fprintf(stderr, "Aplicacion: El archivo con el logo se
abrio correctamente\n");
  }
  else
  {
      fprintf(stderr, "Aplicacion: Hubo un error al abrir el
archivo del logo\n");
  }
    mode_texture();
    pic.setTexture(2001);
  createDisc();
}



//Inicializa los graficos estaticos
void init_static_graphics()
{

}


//Dibuja los graficos dinamicos
void draw_dinamic_graphics()
{
    char buf[256];
  float pos[3];
  pos[0] = pos[1] = pos[2] = 0;

  //Dibujo el metodo que se usa para la desencriptacion
    mode_lines();
  glColor3f(1., 1., 1.);
  pos[0] = -3.5; pos[1] = 1.90 + POS_PILARZ;
  draw_text_line(pos, 0.15, 0.06, "Metodo: Fuerza Bruta");

  //Dibujo n
    pos[0] = -3.5; pos[1] = 1.65 + POS_PILARZ;
  sprintf(buf, "Valor de N: %s",infoWu.n);
  draw_text_line(pos, 0.15, 0.06, buf);

  //Dibujo Busqueda desde
    pos[0] = -3.5; pos[1] = 1.40 + POS_PILARZ;
  sprintf(buf, "Busqueda desde: %s",infoWu.fact_inicial);
  draw_text_line(pos, 0.15, 0.06, buf);
```

```
  //Dibujo busqueda hasta
  pos[0] = -3.5; pos[1] = 1.15 + POS_PILARZ;
  sprintf(buf, "Busqueda hasta: %s",infoWu.fact_final);
  draw_text_line(pos, 0.15, 0.06, buf);

  //Dibujo Busuqeda Actual
  pos[0] = -3.5; pos[1] = 0.9 + POS_PILARZ;
  sprintf(buf, "Verificando numero: %s",&numero_actual[0]);
  draw_text_line(pos, 0.15, 0.06, buf);

  //Dibujo el progreso
  pos[0] = -3.5; pos[1] = 0.65 + POS_PILARZ;
  sprintf(buf, "Progreso: %.2f \%",porcentaje*100);
  draw_text_line(pos, 0.15, 0.06, buf);

  //La barra de porcentaje
    mode_unshaded();
  myProgress.draw(porcentaje);

  /*
  GLfloat color[3] = {1.,1.,1.};
    mode_shaded(color);
  */
  //El disco con el logo de la ESPOL
  mode_texture();
  //glPushMatrix();
         //glRotated(3,0.0,1.0,0.0);
         createMesh();
  //glPopMatrix();
}

//Dibuja los graficos estaticos
void draw_static_graphics()
{
  float pos[3];
  char buf[256];
  pos[0] = pos[1] = pos[2] = 0;
  //Colocamos el titulo principal
  mode_lines();

  glColor3f(0.,0.8,1.);
  pos[0] = -4.2; pos[1] = 8.2 + POS_PILARZ;
  draw_text_line(pos, 0.32, 0.4, "ESCUELA SUPERIOR POLITECNICA
DEL LITORAL");

  glColor3f(1.,1., 1.);
  pos[0] = -3; pos[1] = 7.7 + POS_PILARZ;
  draw_text_line(pos,  0.26,  0.25,  "Plataforma  para  la
Computacion Distribuida");

  //Dibujamos los pilares junto con el texto que lo acompaña
  draw_pillars();

  //Dibujamos la información de la maquina
```

```
  /*INFORMACION QUE POSEE EL aid
        char project_preferences[4096];
        char user_name[256];
        char team_name[256];
        char project_dir[256];
        char boinc_dir[256];
        char wu_name[256];
        char authenticator[256];
        double user_total_credit;
        double user_expavg_credit;
        double team_total_credit;
        double team_expavg_credit;
    */
  mode_lines();
  glColor3f(1., 1., 1.);

  pos[0] = 1.2; pos[1] = 1.90 + POS_PILARZ;
  sprintf(buf, "Usuario/Maquina: %s",aid.user_name);
  draw_text_line(pos, 0.15, 0.06, buf);

  pos[0] = 1.2; pos[1] = 1.65 + POS_PILARZ;
  sprintf(buf, "Division: %s",aid.team_name);
  draw_text_line(pos, 0.15, 0.06, buf);
}

void set_viewpoint(double dist) {
    double x, y, z;
    x = 0;
    y = 3.0*dist;
    z = 11.0*dist;
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(
        x, y, z,  // eye position
        0,-.8,0,      // where we're looking
        0.0, 1.0, 0.      // up is in positive Y direction
    );
}

static void app_init_camera(double dist) {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(
        45.0,       // field of view in degree
        1.0,        // aspect ratio
        1.0,        // Z near clip
        1000.0     // Z far
    );

    set_viewpoint(dist);
}

//Incicializa los graficos que se muestran en el protector de
panatalla
```

```
void init_rsa_graphics() {
   if (GL_NO_ERROR == InitGL()) {
        glClearColor(0.0, 0.0, 0.0, 0.0);
        initlights();

        int viewport[4];
        get_viewport(viewport);
      int w = viewport[2];
      int h = viewport[3];
      app_graphics_resize(w,h);
      init_static_graphics();
      init_dinamic_graphics();
         //app_init_camera(viewpoint_distance);
      //scale_screen(width, height);
    }
}


//Envia a dibujar la tortuga en la pantalla
void dibujar(double time_of_day)
{

  //Borra la pantalla
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  app_init_camera(viewpoint_distance);
  scale_screen(width, height);

  if (angle >= 30 && dir==0)  dir=1;
  if (angle <= -30 && dir==1) dir=0;

  if (angle < 30 && dir==0) {
         angle++;
  } else if (angle>-30 && dir==1) {
         angle--;
         }
  glRotated(angle,0.0,1.0,0.0);

    draw_static_graphics();
  draw_dinamic_graphics();
    glFlush();
  SwapBuffers(hDC);

  /*glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  //glRotated(3,0.0,1.0,0.0);
  //glCallList(1);



  glFlush();
  SwapBuffers(hDC);*/
}
```

G APÉNDICE G: VALORES DE CLAVE RSA Y RANGOS DE BÚSQUEDA.

G.1 Datos para valor de clave RSA N = 12 dígitos.

Valor de N = 700933509247.

Factor primo encontrado= 760531.

Unidades de Trabajo creadas = 5.

Rango #1:	**Rango #2:**	**Rango #3:**
Desde: 1 Hasta: 167443	Desde: 167444 Hasta: 334886	Desde: 334887 Hasta: 502329
Rango #4:	**Rango #5:**	
Desde: 502330 Hasta: 669772	Desde: 669773 Hasta: 837217	

G.2 Datos para valor de clave RSA N=14 dígitos.

Valor de N = 37095613506571.

Factor primo encontrado = 5393053.

Unidades de Trabajo creadas = 5.

Rango #1:	**Rango #2:**	**Rango #3:**
Desde: 1 Hasta: 1218123	Desde: 1218124 Hasta: 2436246	Desde: 2436247 Hasta: 3654369
Rango #4:	**Rango #5:**	
Desde: 3654370 Hasta: 4872492	Desde: 4872493 Hasta: 6090616	

G.3 Tabla de datos del Cálculo de Speedup y Eficiencia.

SpeedUp		
Computadoras	**N=12**	**N=14**
1	0.78	0.75
2	1.44	1.41
3	2.48	2.46
4	2.88	2.81
5	13.65	10.35

Eficiencia		
Computadoras	**N=12**	**N=14**
1	78%	75%
2	72%	71%
3	83%	82%
4	72%	70%
5	273%	207%

REFERENCIAS DE GRÁFICOS

[F1]. **BROWN ROBERT G.**, Engineering a Beowulf-style Compute Cluster, Duke University Physics Department, 2004, 17 p.

[F2]. **PVM OFFICIAL WEB SITE**, <http://www.epm.ornl.gov/pvm/pvmhome.html>, 2003.

[F3]. **LLORENTE MARTÍN IGNACIO**, Necesidad de la Computación Distribuida, Dpto. Arquitectura de Computadores y Automática - Universidad Complutense – Madrid, <http://asds.dacya.ucm.es/nacho/pp_archivos/Necesidad%20Computacion%20Paralela.pdf>, 2003, 5 p.

[F4]. **TANENBAUM ANDREW S.,** Distributed Operating Systems, Prentice-Hall, 1995, 9 p.

[F5]. **CARLSON WILLIAM, EL-GHAZAWI TAREK, NUMRICH ROBERT, YELICK KATHERINE**, Parallel Programming Using A Distributed Shared Memory Model, George Washington University, <ftp://ftp.seas.gwu.edu/pub/upc/downloads/upcsc01.pdf>, 2001, 4 p.

[F6]. **DE CASTRO JUNIOR AMAURY ANTÔNIO**, Modelos Realísticos de Computación Paralela, Universidade Federal de Mato Grosso do Sul, <http://www.dcc.ufla.br/infocomp/artigos/v2.1/amaury.pdf>, 2005, 3 p.

[F7]. **LEMERIE JAN**, "Parallel Systems Course Info, <http://parallel.vub.ac.be>, 2003.

[F8]. **BLAISE BARNEY**, Introduction to Parallel Computing, Lawrence Livermore National Laboratory, <http://www.llnl.gov/computing/tutorials/parallel_comp/#DesignPartitioning>, 2005.

[F9]. **GEIST AL, BEGUELIN ADAM, DONGARRA JACK, JIANG WEICHENG, MANCHEK ROBERT, SUNDERAM VAIDY**, PVM: Parallel Virtual Machine - A Users Guide and Tutorial for Networked Parallel Computing, The MIT Press, <http://www.netlib.org/pvm3/book/pvm_book.html>, 2005, 4 p.

[F10]. **BERNSTEIN PHILIP A**, "Middleware: A Model for Distributed Services", Communications of the ACM 39, 1996, 86-97 p.

[F11]. **CHRISTIAN ULRIK SOTTRUP, JAKOB GREGOR PEDERSEN,** "Developing Distributed Computing SolutionsCombining Grid Computing and Public Computing", Department of Computer Science University of Copenhagen, 2005, 7 p.

[F12]. **IBM RESEARCH**, IBM Research Blue Gene Project Page<http://www.research.ibm.com/bluegene/index.html>.

[F13]. **1000-PENTIUM BEOWULF-STYLE CLUSTER COMPUTER**, <www.genetic-programming.org>, Septiembre 2004

[F14]. **GEIST AL, BEGUELIN ADAM, DONGARRA JACK, JIANG WEICHENG, MANCHEK ROBERT, SUNDERAM VAIDY**, "PVM :

Paralell Virtual Machine A Users' Guide and Tutorial for Networked Paralell Computing", MIT PRESS, 1994, 12 p.

[F15]. **DR.JOHN E. DORBAND, GODDARD SPACE FLIGHT CENTER**, "Beowulf Parallel Linux", < http://ct.gsfc.nasa.gov/annual.reports/ess95contents/sys.beowulf.html >, 2006, 1 p.

[F16]. **"HIGH AVAILABILITY OF LINUX VIRTUAL SERVER",** < http://www.linuxvirtualserver.org/HighAvailability.html >, 2006.

[F17]. **DAVID E. BAKKEN**, "Middleware", Washington State University, Enciclopedia de Computación Distribuida, Press 2003, <http://www.eecs.wsu.edu/~bakken/middleware-article-bakken.pdf>

[F18]. **JAMES RIVER TECHNICAL INC**, Visualization Systems , < http://www.storejrt.com/prismdeskside_bundles.html >, Marzo 2006

[F19]. **SUN FIRE V40Z SERVER,** Sun Microsystems, Inc.<http://store.sun.com/CMTemplate/CEServlet?process=SunStore&cmdViewProduct_CP&catid=116125>, Marzo 2006.

REFERENCIAS BIBLIOGRÁFICAS

[REF.1] **CPUIDLE**, WIKIPEDIA LA ENCICLOPEDIA LIBRE, <http://en.wikipedia.org/wiki/CpuIdle>, Febrero 2006.

[REF.2] **NATALIE AHN, JULIE BLACK, JONATHAN EFFRAT**, “History of Distributed Computing Projects”, Universidad de Stanford - Departamento de Ciencias en Computación, <http://cse.stanford.edu/class/sophomore-college/projects-01/distributed-computing/html/body_history.html>, Marzo 2004.

[REF.3] **“BEOWULF CLUSTERS”,** < http://www.xyroth-enterprises.co.uk/bclust.htm >, Marzo 2004.

[REF.4] **DISTRIBUTED.NET,** Sitio Web Oficial Distributed.net, <http://distributed.net/>, Marzo 2004.

[REF.5] **SETI@home**, Organización Astroseti< http://seti.astroseti.org/setiathome/que.php>, 2003.

[REF.6] **GESIT G.A., KOHL J.A.,** PVM and MPI: a Comparison of Features, P. M. Papadopoulos, 1996, 2 p.

[REF.7] **GESIT G.A., KOHL J.A.,** PVM and MPI: a Comparison of Features, P. M. Papadopoulos, 1996, 3 p.

[REF.8] **EARTH SIMULATOR CENTER,** < http://www.es.jamstec.go.jp/esc/eng/>, 2002.

[REF.9] **TANENBAUM ANDREW S.,** Distributed Operating Systems, Prentice-Hall, 1995, 5 p.

[REF.10] **LI YAOHANG , MASCAGNI MICHAEL**, “Grid-based Monte Carlo Application”, Universidad de Florida Departamento de Computación,<http://www.cs.fsu.edu/~mascagni/papers/RICP2002_3.pdf >, 2002.

[REF.11] .**TANENBAUM ANDREW S.,** Distributed Operating Systems, Prentice-Hall, 1995, 7 p.

[REF.12] **TANENBAUM ANDREW S.,** Distributed Operating Systems, Prentice-Hall, 1995, 8 p.

[REF.13] **CAÑAS JAVIER**, Computación Paralela, Apuntes de Computación Paralela, Universidad Técnica Federico Santa María de Chile, <http://www.inf.utfsm.cl/~jcanas/ComputacionParalela/Apuntes/Capitulo1.pdf>, 2003, 3 p.

[REF.14] **BLAISE BARNEY**, Introduction to Parallel Computing, Lawrence Livermore National Laboratory, <http://www.llnl.gov/computing/tutorials/parallel_comp/#DesignPartitioning>, 2005.

[REF.15] **CATALÁN MIGUEL**, Nuevo modelado de computación paralela con clusters Linux, IV Congreso HISPALinux,

<http://es.tldp.org/Presentaciones/200309hispalinux/16/16.pdf>, 2003, 7 p.

[REF.16] **DEITEL HARVEY M.**, An Introduction to Operating Systems, Addison-Wesley Publishing Company, Segunda Edición, 1990, 29-30 p.

[REF.17] **ARMANDO DE GIUSTI, NAIOUF MARCELO**, Teorías de Sistemas Paralelos, Universidad Nacional de La Plata – Instituto de Investigación en Informática, <http://weblidi.info.unlp.edu.ar/catedras/paralela/Teorias/Clase3-2005.ppt>, 2005.

[REF.18] **GLOSSARY OF HIGH PERFOMANCE COMPUTING TERMS**, Cornell Theory Center - Universidad de Edimburgo, <http://pikachu.tc.cornell.edu/Services/Edu/Topics/Glossary/>, 1996.

[REF.19] **ROMÁN ALONSO GRACIELA, CORTÉS PÉRES ELIZABETH, RUÍZ BARRADA HÉCTOR**, La asignación dinámica de procesos sobre sistemas paralelos, UAM, 2005, 1 p.

[REF.20] **REA ALAN**, An Introduction to Parallel Computing, Queen's University of Belfast - Parallel Computer Centre,

<http://www.pcc.qub.ac.uk/tec/courses/intro/ohp/intro-ohp.html>, 1995.

[REF.21] **CATALÁN MIGUEL**, El manual para el clustering con openMosix, <http://es.tldp.org/Manuales-LuCAS/doc-manual-openMosix-1.0/doc-manual-openMosix-1.0.pdf>, 2004, 34 p.

[REF.22] **KRZYZANOWSKI PAUL**, A taxonomy of Distributed Systems, Universidad de Rutgers, <http://www.pk.org/rutgers/notes/pdf/01-intro.pdf>, 2003, 2-3 p.

[REF.23] **LEMERIE JAN**, Parallel Systems Course Info, Vrije Universiteit Brussel, <http://parallel.vub.ac.be>, 2003.

[REF.24] **TANENBAUM ANDREW S.,** Distributed Operating Systems, Prentice-Hall, 1995, 9 p.

[REF.25] **KRZYZANOWSKI PAUL**, A taxonomy of Distributed Systems, Universidad de Rutgers, <http://www.pk.org/rutgers/notes/pdf/01-intro.pdf>, 2003, 4-6 p.

[REF.26] **ROMO MARCELO**, Procesamiento en Paralelo – Boletín 5, Universidad Internacional del Ecuador – Facultad de Informática y Multimedia,<http://www.internacional.edu.ec/academica/informatica/creatividad/uide-bits/uide-bits-05-2003.pdf >, 2003, 3-4 p.

[REF.27] **PÉREZ DÍAS ARTURO**, Programación Paralela – Modelos de Programación Paralela, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico, Departamento de Ingeniería

Eléctrica - Sección de Computación, <http://www.cs.cinvestav.mx/~adiaz/ParProg2001/ParProg.ps.gz>, 2001, 2-3 p.

[REF.28] **CARLSON WILLIAM, EL-GHAZAWI TAREK, NUMRICH ROBERT, YELICK KATHERINE**, Parallel Programming Using A Distributed Shared Memory Model, George Washington University, <ftp://ftp.seas.gwu.edu/pub/upc/downloads/upcsc01.pdf>, 2001, 4 p.

[REF.29] **LIM K-T**., Mega-Molecular Dynamics on Highly Parallel Computers: Methods and Applications, California Institute of Technology, <http://www.wag.caltech.edu/publications/theses/ktl/Thesis.html>, 1995.

[REF.30] **GARRIDO DANIEL SAMUEL**, Paralelización utilizando Pthreads, PVM y MPI del Problema del Orden Óptimo en la Multiplicación de una Sucesión de Matrices, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico, Departamento de Ingeniería Eléctrica - Sección de Computación, <https://computacion.cs.cinvestav.mx/~sgarrido/cursos/prog_paralela/VPth/ReporteVPthreads.doc>, 2004, 13 p.

[REF.31] **BRADFORD NICHOLS**, Pthreads Programming: A POSIX Standard for Better Multiprocessing, O'Reilly Nutshell, 2005, 6-7 p.

[REF.32] **CHANDRA ROHIT, MENON RAMESH, DAGUM LEO, KOHR DAVID, MANDAN DROR, MCDONALD JEFF**, Parallel Programming in OpenMP, Academia Press - Morgan kaufmann Publishers, 2005, 8 p.

[REF.33] **CROWCROFT JON**, Open Distributed Systems, UCL - LONDON'S GLOBAL UNIVERSITY - Department of Computer Science, <http://www.cs.ucl.ac.uk/staff/J.Crowcroft/ods/ods.html>, 1996.

[REF.34] **PLAZA NIETO EMILIO JOSÉ**, Cluster Heterogéneo De Computadoras, TLDP-ES/LuCAS: Servicios editoriales para la documentación libre en español, <http://es.tldp.org/Manuales-LuCAS/doc-cluster-computadoras/doc-cluster-computadoras-html/node50.html>, 2002.

[REF.35] **FERRARIS LLANOS DIEGO R.**, VSR-COMA: Un Protocolo de Coherencia Caché con Reemplazo para Sistemas Multicomputadores con Gestión de Memoria de tipo COMA, Universidad de Valladolid – Facultad de Informática, 2000, 6 p.

[REF.36] **MARTÍN VICENTE**, Introducción al Paralelismo, Universidad Politécnica de Madrid – Facultad de Informática, <http://www.fi.upm.es/~vicente/tcc/introParalelismo.Herramientas.pdf>, 2005.

[REF.37] **JÁJÁ JOSEPH**. “An Introduction to Parallel Algorithms”, Addison-Wesley, 1992.

[REF.38] **HARRIS TIM J**., A survey of PRAM simulation techniques, ACM Computing Surveys (CSUR), Volume 26 Issue 2, ACM Press, 1994, 188-189 p.

[REF.39] **KESSLER CHRISTOPH W**., The PRAM Programming Language Fork, Linköping University - Department of Computer and Information Science, <http://www.ida.liu.se/~chrke/fork.html#Fork95>, 2005.

[REF.40] **DE CASTRO JUNIOR AMAURY ANTÔNIO**, Modelos Realísticos de Computación Paralela, Universidade Federal de Mato Grosso do Sul, <http://www.dcc.ufla.br/infocomp/artigos/v2.1/amaury.pdf>, 2005, 2 - 3 p.

[REF.41] **VALIANT G**, A bridging model for parallel computation, Communications of the ACM, 33:103-111, 1990.

[REF.42] **JUURLINK BEN H. H., WIJSHOFF HARRY A. G.,** A quantitative comparison of parallel computation models, ACM Press, 1998, 274-275 p.

[REF.43] **ENSLOW P. H**, What is a "distributed" data processing system Computer, 1978.

[REF.44] **COULOURIS GEORGE, DOLLIMORE JEAN, KINDBERG TIM**, Distributed Systems: Concepts and Design, Addison Wesley, Second Edition, 1994.

[REF.45] **BAL H. E.**, Programming Distributed Systems, Prentice Hall, 1990.

[REF.46] **SCHROEDER MICHAEL D**, A estate-of-the-art distributed systems: Computing with BOD, 1-16 p.

[REF.47] **TANENBAUM ANDREW S.**, Distributed Operating Systems, Prentice-Hall, 1995, 2 p.

[REF.48] **MULLENDER S.J.** Introduction in Distributed Systems, 3-18 p.

[REF.49] **ANDERSONY DAVID P.,** Public Computing: Reconnecting People to Science, 2004.

[REF.50] **ECKERSON WAYNE**, "Searching for the Middle Ground", Business Communications Review 25, 1995, 46-50 p.

[REF.51] **SCHREIBER RICHARD**, "Middleware Demystified", Datamation 41, 1995), 41-45 p.

[REF.52] **“SUN HIGH PERFORMANCE COMPUTING”,** Sun Microsystems, Inc., <http://www.sun.com/products/hpc/>, Febrero 2006.

[REF.53] **“PARALLEL COMPUTING INTRODUCTION”,** The National Center for Supercomputing Applications, <http://archive.ncsa.uiuc.edu/SCD/Training/materials/html/intro/> Abril 2004.

[REF.54] **“VISUALIZATION SYSTEMS”,** JAMES RIVER TECHNICAL INC., <http://www.storejrt.com/prismdeskside_bundles.html>, Marzo 2006.

[REF.55] **“SUN FIRE V40Z SERVER”,** Sun Microsystems, Inc.<http://store.sun.com/CMTemplate/CEServlet?process=SunStore&cmdViewProduct_CP&catid=116125>, Marzo 2006.

[REF.56] **“SUPERCOMPUTERS”**, CALLE DAN, <http://ei.cs.vt.edu/~history/SUPERCOM.Calle.html>, Octubre 2003

[REF.57] **“SUPERORDENADORES”,** WIKIPEDIA LA ENCICLOPEDIA LIBRE, <http://es.wikipedia.org/wiki/Supercomputadora>, Marzo 2006.

[REF.58] **“IBM RESEARCH BLUE GENE PROJECT PAGE”,** IBM RESEARCH, <http://www.research.ibm.com/bluegene/index.html>, Marzo 2006.

[REF.59] **“NASA: CREAN SUPERCOMPUTADORA”**, BBC Mundo.com, <http://news.bbc.co.uk/hi/spanish/science/newsid_3543000/3543714.stm>, Agosto 2004.

[REF.60] **“CLUSTER DE COMPUTADORAS”,** WIKIPEDIA LA ENCICLOPEDIA LIBRE, <http://es.wikipedia.org/wiki/Cluster_de_computadores>, Marzo 2006.

[REF.61] **BRIAN R HANLY AND DANIEL KL TUNG**, “Genetic Programming on Clusters”, School of Computer Science and Software

Engineering Monash University, < www.buyya.com/csc433/GeneticProgClusters.pdf>, 2004.

[REF.62] **ORCERO, DAVID SANTO**, “Simulación y optimización paralela de agregados del Silicio”, Disertación de Maestría en Ciencias de la Computación. Departamento de Lenguajes y Ciencias de la Computación. Universidad de Málaga, 2002.

[REF.63] **ARNAIZ, JESÚS**, "Supercomputación y proceso en paralelo", <http://html.rincondelvago.com/supercomputacion-y-proceso-en-paralelo.html>, Enero 2005.

[REF.64] **SHANKLAND, STEPHEN**. “InfiniBand reborn for supercomputing”, <http://news.com.com/2100-1001-966777.html>, Abril 2005.

[REF.65] **MORALES VERONICA, NEGIN SHANA ROSE**, “Our Beowulf Cluster”, < http://maven.smith.edu/~vmorales/parallel.html >, Diciembre 2000.

[REF.66] **OSCAR**, Open Sopurce Group, <http://oscar.openclustergroup.org/> , Marzo 2006.

[REF.67] **DES LIGNERIS, BENOIT ET AL**, “Open Source Cluster Application Resources (OSCAR): design, implementation and interest for the computer scientific community.” OSCAR Symposium, Sherbrooke. Mayo 2003

[REF.68] **JHON WILEY & SONS**, "Tools and Environments For Parallel and Distributed Computing", Jhon Wiley & Sons INC, 2004, 20-21 p.

[REF.69] **GEIST AL, BEGUELIN ADAM, DONGARRA JACK, JIANG WEICHENG, MANCHEK ROBERT, SUNDERAM VAIDY,** “PVM : Paralell Virtual Machine A Users’ Guide and Tutorial for Networked Paralell Computing”, MIT PRESS, 1994, 12 p.

[REF.70] **RYTTER WOJCIECH, PAGOURTZIS ARIS**, “Parallel/Distributed Computing using Parallel Virtual Machine (PVM)” , University of Liverpool-Department of Computer Science, <www.csc.liv.ac.uk/~igor/COMP308/files/Lecture20_21.pdf> Marzo 2006.

[REF.71] **G. A. GEIST, J. A. KOHL, P. M. PAPADOPOULOS**, “PVM and MPI : A Comparision of Features”, Mayo 1996.

[REF.72] **“A CONVERSATION WITH DAVID ANDERSON”,** ACM Queue vol. 3, no. 6 - July/August 2005, <http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=313>

[REF.73] **GONZÁLEZ EMILIO**, “Resumen de BOINC”, <http://boinc.astroseti.org/intro.html>, Diciembre 2003.

[REF.74] **CHRISTIAN ULRIK, JACOB GREGOR,** “Developing Distributed Computing Solutions Combining Grid Computing and Public Computing”, University of Copenhagen <http://www.fatbat.dk/thesis/>, 2005, 6-10 p.

[REF.75] **SITIO WEB RSA LABS**, <http://www.rsasecurity.com/rsalabs/node.asp?id=2092>

[REF.76] **DE REYNA MARTÍNEZ JUAN ARIAS**, “Distribución de los Números Primos: Métodos Elementales”, Universidad de Sevilla - Facultad de Matemáticas, <http://www.pdipas.us.es/a/arias/TAN2003-4/Tema2.pdf>, Marzo 2006.