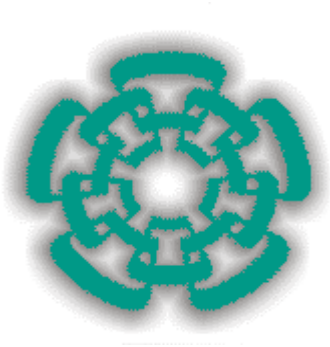


Centro de Investigación y Estudios Avanzados
del Instituto Politécnico Nacional

Departamento de Ingeniería Eléctrica

Sección Computación

Acceso a recursos ejecutables por Internet del Laboratorio de Paralelismo

Tesis que presenta:	**Armando Flores Ibarra**
Optando por el grado de:	**Maestro en Ciencias**
Especialidad:	**Ingeniería Eléctrica**
Opción:	**Computación**

Asesor:

Dr. Arturo Díaz Pérez
Sección de Computación

México, D.F. a 5 de Julio del año 2002

Este trabajo fue parcialmente financiado mediante los proyectos CONACyT 31892A: Algoritmos y arquitecturas de computadoras con dispositivos reconfigurables y CONACYT-CINVESTAV REDII correspondiente al año 2000.

Agradecimientos

A mi madre

Loly Ibarra Caballero

A mis hermanas

Andrea Flores Ibarra

Laura Leticia Flores Ibarra

A mis tíos

Alma Pérez Méndez

Cecilia Ibarra Caballero

Jorge Ibarra Caballero

Karina Bringas Nosti

René Ibarra Caballero

A todos mis familiares por su apoyo y comprensión

A mi asesor

Dr. Arturo Díaz Pérez

A todo el cuerpo de profesores de la Sección de Computación

Contenido

Lista de Figuras

Lista de Tablas

Introducción

Algunos datos indican que actualmente alrededor de 360 millones de personas en todo el mundo tienen acceso a Internet, ya sea desde sus hogares, escuelas o lugar de trabajo incluso existen una gran cantidad de establecimientos públicos que rentan computadoras con dicho acceso. Internet, desde que estuvo al alcance del público en general, ha tenido una enorme aceptación y por lo tanto un gran crecimiento. Tal éxito es debido en gran parte a que es posible tener un acceso remoto a recursos físicamente distantes.
Para este propósito, se han desarrollado una variedad de técnicas para la construcción de aplicaciones Web, las cuales han permitido que la interacción usuario-servidor no sólo se realice de una manera estática, sino que también se les incluya cierto dinamismo. De acuerdo al modelo cliente/servidor, el usuario tiene acceso a recursos, tales como bases de datos, archivos, bancos de información o más recientemente dispositivos específicos, desde los clientes a través de servidores.

Cuando se trata de proveer un mecanismo de acceso remoto a recursos específicos es necesario crear ambientes de trabajo para la ejecución de los programas que controlarán o se comunicarán con tales recursos. Tradicionalmente, esto se había logrado creando *cuentas* para cada uno de los usuarios a quienes se les permitía tal acceso. Sin embargo, esto requiere de un gran trabajo de administración, el cual se multiplica con el número de usuarios a quienes se pretende atender. El crear una *cuenta* involucra definir un ambiente para la ejecución de programas y para acceder a los recursos del sistema. Una de las virtudes de Internet, es ofrecer ambientes de trabajo para un gran número potencial de usuarios sin tener que enterarse de los detalles acerca de cada uno de ellos. En Internet, los recursos están ahí disponibles para quien quiera utilizarlos.

Se han desarrollado ya técnicas y metodologías para consultar bases de datos, bancos de información y bibliotecas digitales, y para realizar comercio electrónico, por citar algunos ejemplos[8][21][25][27]. Sin embargo, cuando se trata de recursos especializados es necesario crear tales ambientes de trabajo *ad hoc*. Tal es el caso de los laboratorios virtuales, en donde se permite tener acceso remoto a las facilidades que ofrece un laboratorio[11].

En este trabajo de tesis se abordará el problema de ofrecer un servicio de acceso remoto al Laboratorio de Paralelismo de la Sección de Computación. Lo que se pretende es brindar a un usuario cualquiera la oportunidad de acceder a los servicios que ofrecen tal tipo de instalaciones desde su lugar de origen y que, al mismo tiempo, éstas pueden ser aprovechadas de una manera más eficiente y estar a la vanguardia del tipo de servicios que hoy en día se ofrecen a través de Internet. Las facilidades mínimas deben permitir ejecutar programas en el laboratorio de paralelismo, proporcionar datos para el procesamiento de los programas y visualizar los resultados.

El presente trabajo de tesis tiene entonces como objetivo general, o nuestro problema a resolver es crear una infraestructura adecuada, tanto en hardware como software, para el acceso a recursos ejecutables por Internet del laboratorio de paralelismo de la Sección de Computación.

Las especificaciones funcionales para nuestro problema son:

- Especificar un protocolo de comunicación entre el cliente y el servidor de recursos ejecutables.
- Especificar la metodología a utilizar para la creación de dichos recursos ejecutables (JSP, ASP, Java Servlets, CGIs, etc...).
- Desarrollo de una interfase funcional entre el servidor de recursos ejecutables y el usuario.
- Construir un catálogo de aplicaciones a ser invocadas remotamente.
- Desarrollar una herramienta de balance de carga para permitir el acceso concurrente de varios usuarios al laboratorio de paralelismo.
- Brindar resultados seguros y confiables al usuario.
- Que sea accesible al público.

De acuerdo a los objetivos específicos anteriores el problema a resolver se puede descomponerse en dos partes principales:

La interfaz:

- Diseño e implementación de las interfaces y protocolos de comunicación necesarios para la comunicación entre cliente-servidor y servidor-laboratorio-paralelo, lo cual permita al usuario poder acceder y hacer uso de los recursos brindados por el Laboratorio Paralelo de una manera remota.

Repartición de recursos:

- Diseño e implementación de un algoritmo de balance de carga el cual le permita al servidor manejar de una manera más eficiente los recursos con los que cuenta el sistema.

La metodología, para poder brindar una solución adecuada y correcta al problema que se plantea es la siguiente (ver Figura I-1):

- Diseño e implementación de una interfase visual entre el cliente y el servidor para el acceso y ejecución de los recursos.
- Diseño e implementación de un protocolo de comunicación entre el servidor Web y el laboratorio paralelo.
- Diseño e implementación de una herramienta de visualización la cual permita al usuario ver de manera gráfica y textual el progreso del recurso seleccionado, así como también las diversas tareas realizadas por el sistema.
- Diseño e implementación de un algoritmo de balance de carga, el cual habilite al sistema con la capacidad de distribuir de una manera eficiente su carga de trabajo.

Los primeros tres puntos resuelven el problema de la interfaz y el último punto resuelve el problema de repartición de recursos.

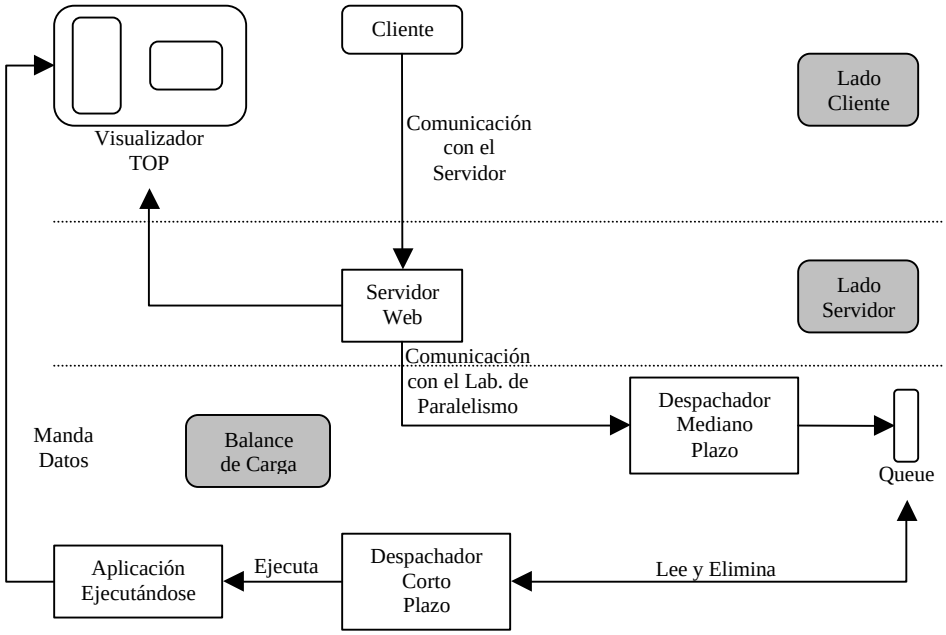


Figura I-1. Pasos a seguir para la resolución del problema.

Finalmente, esta tesis está organizada en 6 capítulos:

En el Capítulo 1 se discuten diversas tecnologías existentes para la creación de sitios Web con contenido dinámico y ejecución remota de recursos.

El Capítulo 2 presenta diversas arquitecturas paralelas las cuales pueden tomarse como base para la implementación de un laboratorio de paralelismo, las herramientas disponibles para la creación de programas paralelos utilizando el paradigma de paso de mensajes, y algunos resultados obtenidos de pruebas de comunicación realizadas en el laboratorio de paralelismo de la sección.

El Capítulo 3 muestra la manera de invocar remotamente los recursos del laboratorio de paralelismo utilizando CGI's y la generación dinámica de tablas y gráficas de los resultados obtenidos al ejecutar dichos recursos.

El Capítulo 4 aborda el tema de la calendarización de procesos. Se describe cómo se puede utilizar eficientemente los recursos con los que cuenta el laboratorio de paralelismo al calendarizar de manera adecuada la ejecución de las diversas tareas que pueden ser solicitadas por uno o varios usuarios a la vez.

El Capítulo 5 habla acerca de las facilidades visuales típicas que deben ofrecerse al usuario cuando requiere de la ejecución de uno de los recursos del sistema. Estas incluyen una herramienta que muestre el avance que la aplicación paralela vaya teniendo y una herramienta que muestre las actividades realizadas por el sistema durante la ejecución del recurso seleccionado por el usuario.

El Capítulo 6 muestra algunos resultados obtenidos de la ejecución de las aplicaciones paralelas con las que cuenta el sistema, se describe también brevemente cada una de ellas.

Capítulo 1

Tecnologías para la Construcción de Aplicaciones Cliente/Servidor en Internet

En un principio el uso de Internet se limitó a la búsqueda de información estática, el usuario no tenía la oportunidad de interactuar con el sitio web. Conforme la tecnología ha avanzando y el acceso a Internet se ha popularizado, han aparecido nuevas necesidades por parte de los usuarios. Actualmente, la mayoría de las aplicaciones desarrolladas para Internet permiten interactuar con información dinámica, que se genera en el momento del acceso. La mayoría de las aplicaciones modernas para Internet trabajan de acuerdo al modelo cliente/servidor, en el cual las aplicaciones cliente solicitan servicios a procesos servidor. El cliente y el servidor requieren conocer ciertas convenciones bien definidas antes de que los servicios puedan ser proporcionados (y aceptados). Este conjunto de condiciones se agrupa en un protocolo de intercambio de información el cual debe de ser implementado en ambos lados de la conexión.

En este capítulo se discuten las tendencias en la construcción de aplicaciones dinámicas para Internet. En la primera parte se discuten las metodologías a considerar en la ingeniería de software para construir una aplicación cliente/servidor. En la segunda parte se presenta algunas de las técnicas más usuales para la construcción de páginas web dinámicas. Finalmente, en la tercera parte se discuten algunas aplicaciones no convencionales que se benefician de la integración del modelo cliente/servidor con el uso de páginas dinámicas.

1.1 Ingeniería de los Sistemas Cliente/Servidor

Cuando se va a desarrollar un sistema basado en computadoras, el ingeniero se ve limitado por las restricciones impuestas por la tecnología existente, su situación mejora conforme las nuevas tecnologías ofrecen capacidades que no estuvieron disponibles para los ingenieros anteriores. A principios del siglo XX, el desarrollo de una nueva generación de máquinas herramientas capaces de mantener tolerancias sumamente reducidas permitió a los ingenieros diseñar un nuevo proceso de fabricación denominado producción en masa.

La evolución de los sistemas distribuidos, por otra parte, ha permitido desarrollar nuevos enfoques acerca de la forma en que se realiza el trabajo y acerca de la forma en que se procesa la información en el seno de una organización. Las nuevas estructuras organizativas y los nuevos enfoques de procesamiento de información (p. ej. sistemas de apoyo de decisiones, software para trabajo en grupos y gestión de gráficos) representan un esquema radicalmente diferente con respecto a tecnologías anteriores, basadas en grandes computadoras y minicomputadoras.

Las tecnologías de hardware, de software, de bases de datos y de redes contribuyen todas ellas al desarrollo de sistemas distribuidos y cooperativos. En su forma más general, un sistema distribuido y cooperativo se presenta como en la Figura 1.1: Un sistema raíz, que típicamente será un servidor central, actúa como un depósito de datos corporativos; el sistema raíz esta conectado con servidores secundarios (que típicamente son estaciones de trabajo potentes, o PC) y que poseen el doble papel de actualizar y solicitar los datos corporativos que están mantenidos por el sistema raíz[25].

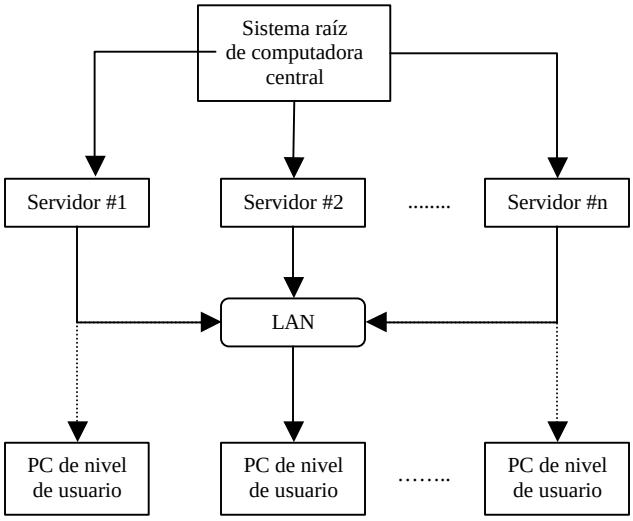


Figura 1.1 Arquitecturas de sistemas distribuidos y cooperativos en un entorno corporativo.

Los servidores secundarios se utilizan en sistemas departamentales locales y desempeñan un papel clave al interconectar todas las computadoras de nivel de usuario a través de una red de área local (LAN).

La mayoría de las aplicaciones modernas para Internet trabajan de acuerdo al modelo cliente/servidor (C/S). En una estructura C/S, la computadora que reside por encima de otra computadora (en la Figura 1.1) se denomina servidor, y las computadoras de nivel inferior se denominan clientes. Los clientes solicitan servicios, y el servidor los proporciona. Sin embargo, en el contexto de la arquitectura de sistemas distribuidos representada en la Figura 1.2, se pueden llevar a cabo un cierto número de implementaciones distintas:

- **Servidores de archivos.** El cliente solicita registros específicos de un archivo. El servidor transmite estos registros al cliente a través de la red.
- **Servidores de bases de datos.** El cliente envía solicitudes en *lenguaje de consulta estructurado* (SQL) al servidor. Éstas se trasmiten como mensajes a través de la red. El servidor procesa la solicitud SQL y halla la información solicitada, pasando únicamente los resultados al cliente.
- **Servidores de transacciones.** El cliente envía una solicitud que invoca procedimientos remotos en el centro servidor. Los *procedimientos remotos* pueden ser un conjunto de sentencias SQL. Se produce una transacción cuando una solicitud da lugar a la ejecución de procedimientos remotos y a la transmisión del resultado devuelto al cliente.
- **Servidores de grupos de trabajo.** Cuando el servidor proporciona un conjunto de aplicaciones que hacen posible la comunicación entre clientes (y entre personas que los usan) mediante el uso de texto, imágenes, boletines electrónicos, vídeo y otras representaciones.

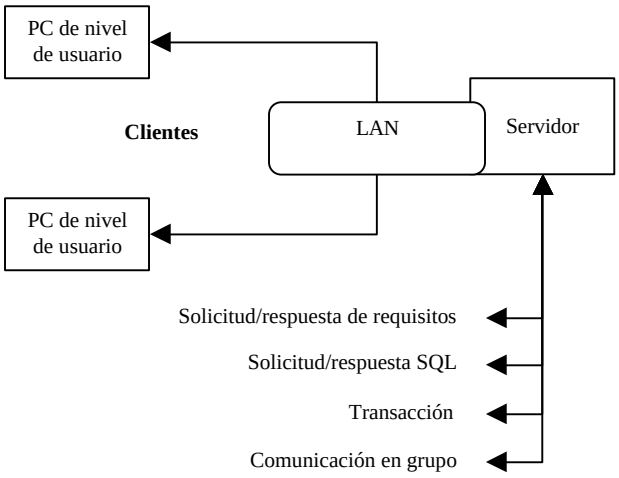


Figura 1.2 Opciones de la arquitectura cliente/servidor.

El software para una arquitectura Cliente/Servidor no debe visualizarse como una aplicación monolítica que deberá implementarse en una máquina, en vez de ello debe poseer varios componentes distintos que se pueden asociar al cliente o al servidor, o se pueden distribuir entre ambas máquinas. Entre las componentes más importantes de una aplicación C/S se pueden mencionar las siguientes:

- **Componentes de interacción con el usuario y presentación.** Este componente implementa todas las funciones que típicamente se asocian a una interfaz gráfica de usuario (GUI por sus siglas en inglés).
- **Componentes de aplicación.** Este componente implementa los requisitos definidos por la aplicación. Por ejemplo, una aplicación de negocios podría producir toda una gama de informes impresos basados en entradas numéricas, cálculos, información de una base de datos, y otros aspectos. Una aplicación para trabajo en grupo podría proporcionar las capacidades adecuadas para hacer posible la comunicación mediante un boletín electrónico o correo electrónico. En ambos casos, el software de aplicación se puede descomponer de tal modo que alguno de los componentes residan en el cliente y otros residan en el servidor.
- **Gestión de bases de datos.** Este componente lleva a cabo la manipulación y gestión de datos requerida por una aplicación. La manipulación y gestión de datos puede ser tan sencilla como la transferencia de un registro, o tan compleja como el procesamiento de transacciones sofisticadas en SQL.

Además de estos componentes, existe en todos los sistemas C/S otro bloque de construcción del software el cual suele denominarse software intermedio. El *software intermedio* consta de elementos de software que existen tanto en el cliente como en el servidor, e incluye elementos de sistemas operativos en red así como un software de aplicación especializado que presta su apoyo a las aplicaciones específicas de bases de datos, a estándares de distribución de solicitudes de objetos, a tecnologías de trabajo en grupo, a gestión de comunicaciones, y a otras características que facilitan la conexión cliente/servidor[25].

Una vez que se han determinado los requisitos básicos de una aplicación C/S, el ingeniero de software debe de decidir la forma en que distribuirá los componentes de software descritos anteriormente entre el cliente y el servidor. Cuando la mayor parte de la funcionalidad asociada a cada uno de los tres componentes (componentes de interacción con el usuario y presentación, componentes de aplicación, gestión de bases de datos) se le asocia al servidor, se ha creado un diseño de aplicación enfocado a *servidor principal*. Por el contrario, cuando el cliente implementa la mayor parte de los componentes de interacción/presentación con el usuario, de aplicación y de bases de datos, se tiene un diseño de aplicación enfocado a *cliente principal*.

Los clientes principales suelen encontrarse cuando se implementan arquitecturas de servidor de archivos y de servidor de base de datos. En este caso, el servidor proporciona apoyo para la gestión de datos, pero todo el software de aplicación y de GUI reside en el cliente. Los servidores principales son los que suelen diseñarse cuando se implementan sistemas de transacciones y de trabajo en grupo. El servidor proporciona el apoyo de la aplicación necesario para responder a transacciones y comunicaciones que provengan de los clientes. El software del cliente se centra en la gestión de GUI y de comunicaciones.

Se pueden utilizar clientes principales y servidores principales para ilustrar el enfoque general de asignación de componentes de software de cliente/servidor. Sin embargo, un enfoque más granular para la asignación de componentes de software permite aplicar cinco configuraciones diferentes:

1. **Presentación distribuida.** En este enfoque cliente/servidor rudimentario, la lógica de la base de datos y la lógica de las aplicaciones permanece en el servidor, típicamente una computadora central. El servidor contiene también la lógica para preparar información de pantalla, empleando un software tal como CICS[1]. Se utiliza un software especial basado en PC para transformar la información de pantalla basada en caracteres que se transmite desde el servidor en una presentación GUI en un PC.
2. **Presentación remota.** En esta extensión del enfoque de presentación distribuida, la lógica primaria de la base de datos y de la aplicación permanecen en el servidor, y los datos enviados por el servidor serán utilizados por el cliente para preparar la presentación del cliente.
3. **Lógica distribuida.** Se asignan al cliente todas las tareas de presentación del usuario y también todos los procesos asociados a la introducción de datos tales como la validación de nivel de campo, la formulación de consultas de servidor, y las solicitudes de informaciones de actualizaciones del servidor. Se asignan al servidor las tareas de gestión de la base de datos, y los procesos para las consultas del cliente, para actualizaciones de archivos del servidor, para control de versión de clientes, y para aplicaciones de ámbito general de la empresa.
4. **Gestión de datos remota.** Las aplicaciones del servidor crean una nueva fuente de datos dando formato a los datos que se han extraído de algún otro lugar. Las aplicaciones asignadas al cliente se utilizan para explotar los nuevos datos a los que se ha dado formato mediante el servidor. En esta categoría se incluyen los sistemas de apoyo de decisiones.
5. **Bases de datos distribuidas.** Los datos de los cuales consta la base de datos se distribuyen entre múltiples clientes y servidores. Consecuentemente, el cliente debe de admitir componentes de software de gestión de datos así como componentes de aplicación y de GUI.

[1] (http://www-3.ibm.com/software/ts/cics/)

Aun cuando no existen reglas absolutas que describan la distribución de componentes de aplicaciones entre el cliente y el servidor, suelen seguirse las siguientes líneas generales:

- *El componente de presentación/interacción suele ubicarse en el cliente*. La disponibilidad de entornos basados en ventanas y de la potencia de cómputo necesaria para una interfaz gráfica de usuario hace que este enfoque sea eficiente en términos de costo.
- *Si es necesario compartir la base de datos entre múltiples usuarios conectados a través de LAN, entonces la base de datos suele ubicarse en el servidor*. El sistema de gestión de la base de datos y la capacidad de acceso a la base de datos también se asignan al servidor, junto con la base de datos física.
- *Los datos estáticos que se utilicen como referencia deberían de asignarse al cliente*. Esto sitúa los datos más próximos al usuario quien tiene necesidad de ellos, y minimiza un tráfico de red innecesario y la carga del servidor.

La decisión final acerca de la distribución de componentes debería de estar basada no solamente en la aplicación individual sino en la mezcla de aplicaciones que estén funcionando en el sistema.

Se utiliza toda una gama de mecanismos distintos para enlazar los distintos componentes de la arquitectura cliente/servidor. Estos mecanismos están incluidos en la estructura de la red y del sistema operativo, y resultan transparentes para el usuario. Los tipos más comunes de mecanismos de enlazado son:

- **Tuberías (pipes)**: se utilizan mucho en los sistemas basados en UNIX; las tuberías permiten la mensajería entre distintas máquinas que funcionen con distintos sistemas operativos.
- **Llamadas a procedimientos remotos**: permiten que un proceso invoque la ejecución de otro proceso o módulo que resida en una máquina distinta.
- **Interacción cliente/servidor SQL**: se utiliza para pasar solicitudes SQL y datos asociados de un componente (típicamente situado en el cliente) a otro componente.

Los sistemas cliente/servidor se desarrollan empleando las actividades de ingeniería del software clásico: análisis, diseño, construcción y depuración a medida. Lo anterior permite que un sistema C/S evolucione a partir de un conjunto de requisitos generales para llegar a ser una colección de componentes de software ya validados que han sido implementados en máquinas cliente y servidor.

En general, la comprobación (pruebas) de software de cliente/servidor se produce en tres niveles diferentes: (1) Las aplicaciones del cliente individuales se comprueban de modo *desconectado* (el funcionamiento del servidor y de la red subyacente no se consideran); (2) las aplicaciones de software de cliente y del servidor asociado se prueban al unísono, pero no se ejercitan específicamente las operaciones de red; (3) se comprueba la arquitectura completa de C/S, incluyendo el rendimiento y funcionamiento de la red.

Aun cuando se efectúen muchas clases distintas de pruebas en cada uno de los niveles de detalle anteriores, es frecuente encontrar los siguientes enfoques de comprobación para aplicaciones C/S:

- **Comprobaciones de función de aplicación.** En esencia, la aplicación se comprueba en solitario en un intento de descubrir errores de su funcionamiento.
- **Comprobaciones de servidor.** Se comprueban la coordinación y las funciones de gestión de datos del servidor. También se considera el rendimiento del servidor (tiempo de respuesta y traspase de datos en general).
- **Comprobaciones de transacciones.** Se crea una serie de comprobaciones adecuada para comprobar que todas las clases de transacciones se procesen de acuerdo con los requisitos. Las comprobaciones hacen especial hincapié en la corrección de procesamiento, y también en los temas de rendimiento.
- **Comprobaciones de comunicaciones a través de la red.** Estas comprobaciones verifican que la comunicación entre los nudos de la red se produzca correctamente, y que el paso de mensaje, las transacciones y el tráfico de red relacionado tenga lugar sin errores.

Para llevar a cabo estos enfoques de comprobación, se recomienda el desarrollo de perfiles operacionales derivados de escenarios cliente/servidor. Un *perfil operacional* indica la forma en que los distintos tipos de usuarios interactúan con el sistema C/S. Estos es, los perfiles proporcionan una trama de utilización que se puede aplicar cuando se diseñan y ejecutan las comprobaciones. Por ejemplo, para un determinado tipo de usuarios la trama puede incluir ¿qué porcentaje de las transacciones serán consultadas?, ¿cuántas actualizaciones serán realizadas? y ¿qué peticiones serán efectuadas?

Para desarrollar el perfil operativo, es preciso derivar un conjunto de *escenarios de usuario*. Cada escenario debe considerar quién es el usuario, dónde (en la arquitectura física) se produce la interacción con el sistema, cuál es la transacción, y porqué ha sucedido.

1.2 Tendencias en el diseño de Páginas Web

Sin duda un tipo importante de aplicaciones C/S en la actualidad es aquella que contiene a las que trabajan sobre Internet. La manera usual de organizarlas es simple, mediante páginas web solicitadas por un navegador a un servidor web. Sin embargo, en las aplicaciones modernas no es suficiente con poder acceder a información y navegar a través de ella, se requiere que se pueda acceder a otros servicios como por ejemplo consulta a bases de datos, realización de transacciones y acceso a servicios específicos. El mecanismo básico para la construcción de aplicaciones de tal tipo requiere la facilidad de creación de **páginas web dinámicas**. Mediante las páginas dinámicas un usuario tiene la capacidad de interactuar con el sitio web y con los diferentes **recursos** que éste pueda ofrecer. Se han desarrollado diversas tecnologías para la creación de páginas web dinámicas y aplicaciones remotas ejecutables. En esta sección se describen cuatro tecnologías básicas para la construcción de páginas dinámicas, no porque sean la**s** únicas disponibles sino por que ellas facilitan el entendimiento de la solución al problema planteado en esta tesis.

1.2.1 CGI: Common Gateway Interface

La especificación de la Interfaz de Puerta Común (CGI por sus siglas en inglés) permite a los servidores Web ejecutar otros programas e incorporar su salida en texto, gráficos, y audio enviado a un navegador Web. El servidor y el programa CGI trabajan juntos para mejorar y ajustar las capacidades del World Wide Web. Proporcionando una interfaz estándar, la especificación CGI permite a los desarrolladores usar una gran variedad de herramientas de programación. Los programas CGI pueden ser usados para realizar el trabajo detrás del procesamiento de formas, buscando registros en una base de datos, mandando correos electrónicos, construyendo en la marcha contadores de página y varias otras actividades.

Los navegadores (browsers) y los servidores Web se comunican utilizando el Protocolo de Transferencia de Hipertexto (http) y mediante URL's (Universal Resource Locator). Técnicamente, un URL es una forma del Identificador de Recursos Universales (URI) usado para acceder un objeto usando protocolos de Internet existentes. De una forma simplificada, seis cosas suceden normalmente cuando desde un navegador web se visita a un sitio en la WWW:

1. El navegador decodifica la primera parte del URL y contacta al servidor.
2. El navegador proporciona lo demás del URL al servidor.
3. El servidor traduce el URL en una trayectoria y un nombre de archivo.
4. El servidor envía el documento al navegador.
5. El servidor rompe la conexión.
6. El navegador despliega el documento.

El navegador Web no conoce mucho acerca de los datos que requiere el documento. El sólo presenta la URL y encuentra lo que obtiene cuando se le regresa una respuesta. El servidor proporciona cierto código, utilizando especificaciones MIME, para decirle al navegador el contenido de la página. La mayoría de los documentos Web son HTML: solo texto plano con instrucciones incrustadas para formateo y despliegue.

Por sí mismo, el servidor solamente tiene la capacidad para mandar documentos y decirle al navegador qué tipos de documentos son. Sin embargo, los servidores actuales también tienen la posibilidad de invocar programas adicionales.

Cuando un servidor ve que un URL apunta a un archivo, manda de vuelta el contenido de ese archivo. Cuando un URL apunta a un programa, sin embargo, el servidor activa el programa. El servidor entonces envía de vuelta la salida del programa como si fuera un archivo. Un programa CGI puede leer y escribir archivos de datos, un servidor Web solo puede leerlos. Así, un programa CGI puede producir diferentes resultados cada vez que es ejecutado, como se muestra en la siguiente Figura 1.3.

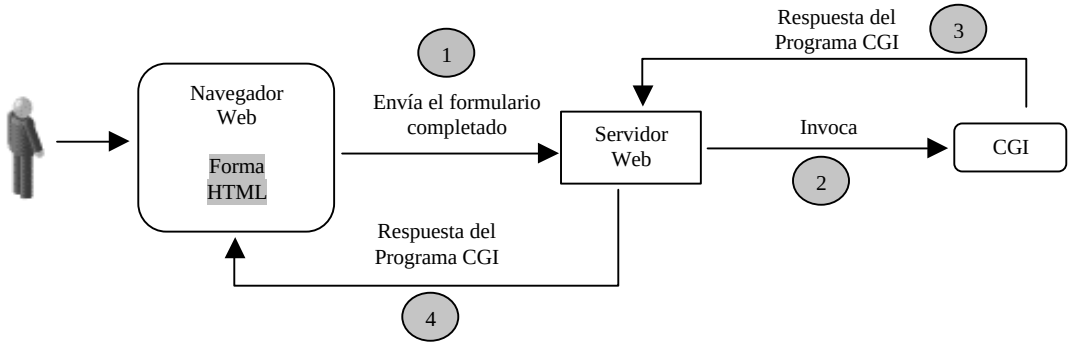


Figura 1.3. Invocación de un programa CGI por un usuario remoto.

Los programas CGI son comúnmente llamados scripts porque los primeros programas fueron escritos usando scripts de shell de **UNIX** (sh ó csh) o mediante el lenguaje Perl[14]. Cuando uno ejecuta un programa Perl, las instrucciones Perl son interpretadas y compiladas en instrucciones de máquina. Otros lenguajes, como C, son compilados antes, y el ejecutable resultante normalmente no es llamado un script sino porgrama. En el contexto de CGI, sin embargo, los programas compilados e interpretados son llamados scripts. Los programas CGI pueden residir en cualquier parte del sistema de archivos; el servidor es configurado para poder localizarlos.

El manejo de entradas de forma es uno de los usos más comunes de los scripts CGI hoy en día. Una forma es sólo un grupo de etiquetas (tags) HTML que generan tales elementos como campos de entrada, listas de cajas (list boxes), cajas marcables (check boxes), botones radio (radio buttons) y botones apretables (push buttons). Las formas permiten al usuario ver su página Web para interactuar con su sitio Web dando información o haciendo una selección.

1.2.1.1 CGI y Bases de Datos

Uno de los usos primarios de la tecnología Web es la de proveer una interfaz (la cual sea independiente de la plataforma que se utilice) para datos almacenados en bases de datos. Los CGI tiene las características necesarias para la creación de aplicaciones Web que operen sobre bases de datos.

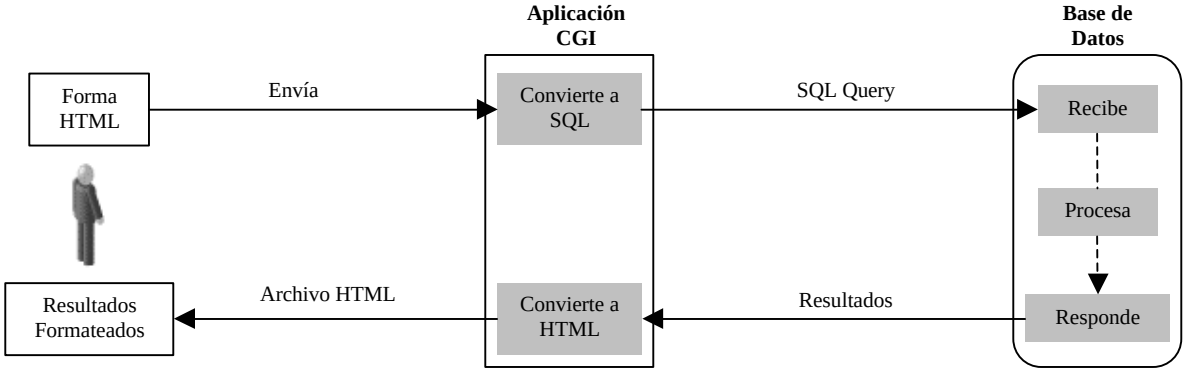


Figura 1.4 Acceso a una base de datos utilizando CGIs.

La Figura 1.4 muestra de forma general como un programa CGI puede ser utilizado para trabajar con bases de datos utilizando un navegador Web como interfaz principal. El lenguaje Perl ofrece una buena alternativa para la creación de este tipo de aplicaciones, ya que cuenta con módulos como DBI[22][34] el cual provee una interfaz común entre Perl y varias máquinas de bases de datos.

La programación de scripts CGI ofrece ventajas como una excelente portabilidad de código entre sistemas operativos al utilizar un lenguaje como Perl, además Perl por ser código-abierto su funcionalidad (como conectores a bases de datos, funciones de encriptación) siempre esta en continuo crecimiento. La gran mayoría de los servidores Web actuales ofrecen la posibilidad de ejecutar scripts CGI. Por otro lado, el uso de CGIs tiene ciertas desventajas, el programador debe poseer un cierto nivel de programación para explotar de manera adecuada todas sus características y dar mantenimiento a la aplicación, ya que conforme esta crece, su complejidad también lo hace y sería difícil para un programador principiante llevar a cabo este mantenimiento [22].

1.2.2 Java Servlets

Un servlet es un programa que reside en un servidor, escrito en código Java, que administra mensajes entre un cliente y un servidor. El Java Servlet API define una interfaz estándar para la solicitud y respuesta de mensajes de tal manera que los servlets puedan ser portables entre plataformas y entre diferentes servidores Web de aplicaciones. Los servlets pueden responder a solicitudes de cliente mediante la construcción dinámica de una respuesta la cual es devuelta al cliente. Un solo servlet puede ser invocado múltiples veces para servir solicitudes de múltiples clientes. Los servlets pueden enviar solicitudes a otros servidores y servlets.

Un servlet se ejecuta dentro de un servidor de aplicaciones. Los servidores de aplicaciones son un tipo especial de servidores Web; extienden las capacidades de un servidor Web para manejar peticiones para servlets y otras aplicaciones Web creadas con algún otro lenguaje. El servidor por sí mismo carga, ejecuta, y administra los servlets.

El servidor utiliza un intérprete Java para ejecutar programas Java; el cual es conocido como Máquina Virtual de Java (JVM por sus siglas en inglés).

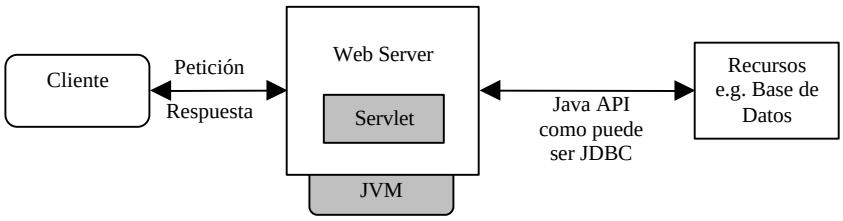


Figura 1.5 Proceso Java Servlets

El flujo básico para la invocación de un servlet se ilustra en la Figura 1.5 y se describe a continuación:

1. El cliente envía una solicitud al servidor.
2. El servidor crea una instancia (carga) del servlet y crea un thread para el proceso servlet. El servlet se ejecuta a través de la JVM.
3. El servidor envía la información de la solicitud al servlet.
4. El servlet construye una respuesta y se la envía al servidor.
5. El servidor envía la respuesta de vuelta al cliente.

El servlet construye dinámicamente la respuesta utilizando información proveniente de la solicitud del cliente, además de datos obtenidos de otras fuentes si es necesario. Tales fuentes pueden ser otros servlets, objetos compartidos, archivos de recursos, y bases de datos[23][10].

Java Servlets ofrece ventajas como el de ser portable entre varios servidores Web y plataformas ya que utiliza Java como lenguaje de programación. El de ser eficiente ya que cuando el cliente solicita que un mismos servlet sea ejecutado el servidor Web solamente necesita crearlo y cargarlo una vez, en comparación a los CGI que necesitan ser cargados cada vez que son invocados lo cual representa una desventaja en desempeño. Por otro lado, puede representar una desventaja para un programador sin experiencia en Java y programación orientada a objetos, ya que su aprendizaje puede dificultársele. Otra desventaja es que el código de los Servlets puede llegar a ser difícil de leer y sobre todo de modificar.

1.2.3 Active Server Pages

Active Server Pages es un marco de trabajo independiente del lenguaje (como JavaScript y VBScript) diseñado por Microsoft® para una codificación eficiente de servidores de scripts. Estos son diseñados para ser ejecutados en un servidor Web en respuesta a una petición de usuario para un URL [20]. ASP esta actualmente disponible para los siguientes servidores Web:

- Microsoft Internet Information Server (IIS) 3.0
- Microsoft Internet Information Server (IIS) 4.0
- Microsoft Personal Web Server (PWS)
- O'Reilly WebSite Pro

El proceso que se realiza al momento de acceder a una página ASP lo podemos resumir en 8 pasos principales (como se ilustra en la Figura 1.6):

1. El usuario teclea un URL a través del navegador Web y se conecta al servidor requerido.
2. El servidor obtiene el archivo HTML requerido.
3. El servidor regresa al usuario el archivo HTML requerido el cual contiene una forma para ser llenada.
4. El usuario llena la forma y presiona el botón SUBMIT (Presentar) con lo cual la información es enviada de regreso al servidor la cual será utilizada por el programa ASP cuyo nombre se encuentra empotrado en el archivo HTML.
5. El servidor corre el archivo ASP requerido dándole la información que fue enviada en la forma llenada por el usuario.
6. Si el archivo ASP requiere información de una base de datos, se conecta a ella y recupera los datos.
7. El archivo ASP genera una página HTML que contiene todos los resultados.
8. El servidor envía dicha página de resultados de regreso al navegador Web del cliente.

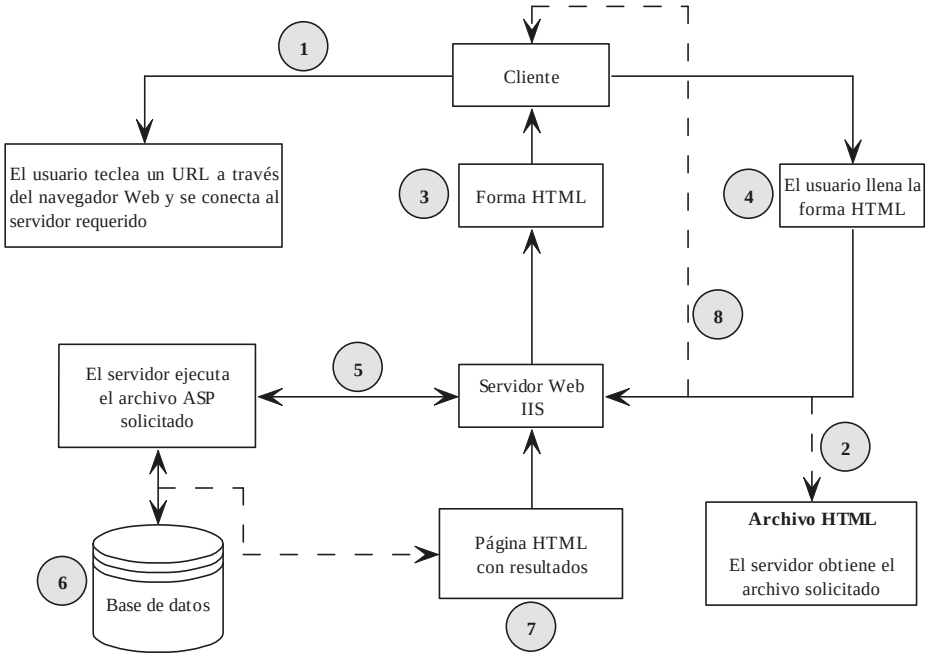


Figura 1.6 Proceso de ejecución de una página ASP.

1.2.3.1 ASP y Bases de Datos

ASP utiliza una tecnología llamada **ActiveX Data Objetcs** (ADO) para trabajar con bases de datos. ADO es una tecnología ActiveX la cual se encuentra incluida dentro del **IIS** (Internet Information Server). ADO abarca un número de objetos los cuales pueden ser utilizados, de los cuales los 3 más importantes son: **Command Object**, **Connection Object** y **Recordset Object**. Siempre que se realice una operación de base de datos, estos 3 objetos estarán presentes[20].

Quizás la característica más importante de ASP es la de utilizar componentes programables. Uno puede crear estos componentes utilizando herramientas como Visual Basic, Visual C++, Visual J++, Borland Delphi, y Powersoft PowerBuilder. Así, se pueden integrar aplicaciones Web con sistemas existentes cliente-servidor. ASP tiene 2 limitaciones principales: solamente puede ejecutarse en plataformas Microsoft utilizando IIS como servidor Web de aplicaciones, y su uso requiere de experiencia y habilidades de programación[22].

1.2.4 JavaServer Pages (JSP)

JSP, o Java Server Pages, es una tecnología desarrollada por Sun Microsystems la cual permite una fácil creación y mantenimiento de servidores de páginas HTML, el cual puede ser usado tanto como un tipo de HTML dinámico y como reemplazo de scripts CGI. Conceptualmente JSP es similar a ASP ya que ambos proveen seguimiento de sesión e interacciones sofisticadas con bases de datos.

Sin embargo, los scripts JSP son construidos en Java, el cual es un lenguaje mucho más avanzado que Visual Basic, tiene un mejor soporte para encontrar errores y permite programación concurrente con múltiples hilos (multithreading). También, JSP corre en otras plataformas diferentes a IIS. Un JSP es totalmente compilado hacia un servlet. Así, un JSP es un servlet en cierto sentido; sin embargo, JSP puede ser más fácilmente desplegado, a partir de que pueden residir en cualquier directorio[10].

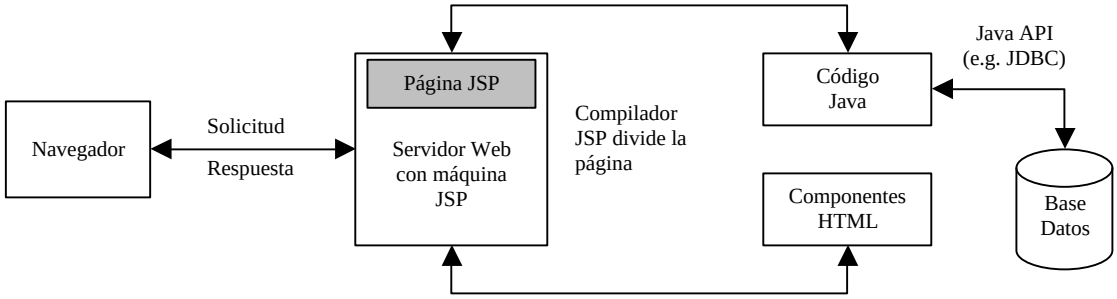


Figura 1.7 Proceso JSP.

El flujo básico para el procesamiento de una página JSP es el siguiente (como se ilustra en la Figura 1.7):

1. Un navegador realiza una solicitud a un sitio Web JSP. Si la página solicitada tiene una extensión ".jsp" entonces el servidor Web pasa la solicitud a la máquina JSP/Servlet.

2. La máquina JSP verifica si la página necesita ser recompilada observando la última fecha de modificación y comparándola con la de la última compilación (si es que existe).
3. El compilador JSP divide la página en sus componentes HTML y su código Java. Generando un nuevo servlet dinámico, y almacenando el código HTML como cadenas de texto en un archivo.
4. La página compilada tiene acceso a la estructura de clases de Java presentes en la Máquina Virtual de Java (JVM por sus siglas en inglés).
5. La página compilada utiliza una clase JSP Writer para enviar el código HTML.

JSP al igual que Java Servlets ofrece independencia entre plataformas y servidores Web ya que también utiliza Java como su lenguaje de programación, estas dos características ofrecen una ventaja inmediata sobre ASP. JSP es además más fácil de codificar que los Java Servlets, ya que permite separa de manera muy clara el código HTML del código Java lo cual hace más fácil el mantenimiento de alguna aplicación creada con esta herramienta. Al igual que los Java Servlets el aprendizaje de JSP puede llegar a ser difícil de aprender para un programador principiante que no tenga conocimientos básicos de Java y Programación Orientada a Objetos.

1.2.4.1 Java Database Connectivity (JDBC)

Se estima que la mitad del desarrollo del software involucra operaciones cliente/servidor. Una de las grandes características de Java ha sido la de poder construir aplicaciones cliente/servidor para bases de datos independientes de la plataforma.

La tecnología JDBC es un API la cual permite un acceso virtual a cualquier archivos de datos tabular. Provee conectividad **DBMS** (Database Management System) para un amplio rango de bases de datos SQL[2].

[2] (http://java.sun.com)

JDBC es un API de Java independiente de la plataforma, de tal manera que no es necesario preocuparse acerca de la máquina de base de datos que se utilice (Oracle, Microsoft SQL Server, etc.) durante la programación. Para proveer esta independencia, JDBC provee un *administrador de manejadores* (driver manager) que dinámicamente mantiene todos los objetos de manejo (driver objects) que las diferentes máquinas de bases de datos necesitan para hacer sus consultas [10].

1.2.5 PHP

PHP es un lenguaje interpretado (por medio de scripts) empotrado en HTML. Mucha de su sintaxis es heredada de C, Java y Perl con algunas características específicas de PHP. La meta del lenguaje es permitir a los desarrolladores escribir en forma rápida páginas generadas dinámicamente.

El nombre PHP viene de PHP: Hypertext Preprocessor. Esto confunde a mucha gente porque la primera letra del acrónimo es el acrónimo. Algún curioso puede visitar Free On-Line Dictionary of Computing para más información sobre acrónimos recursivos.

Si consideramos a PHP el mejor lenguaje para programación WEB, ¿qué se puede decir sobre los otros lenguajes?

1. ¿PHP vs. ASP?
2. ¿PHP vs. Cold Fusion?
3. ¿PHP vs. Perl?

1.2.5.1 PHP vs. ASP?

ASP no es realmente un lengueje en sí mismo, ASP es una acrónimo para Active Server Pages, el lenguaje usado para programar en ASP es Visual Basic Script o JScript. La desventaja de ASP es que es un sistema propietario es es usado nativamente solamente por Internet Information Server (ISS) de Microsoft. Esto limita su disponibilidad a servidores basados en Win32.

Existen varios proyectos que permiten a ASP ejecutarse en otro ambientes y servidores web: InstantASP de la empresa Halcyon (comercial), Chili!Soft ASP de Chili!Soft (comercial) y OpenASP de ActiveScripting.org (libre). Se dice que ASP es más lento y un lenguaje más voluminoso que PHP, y también menos estable. Algunos de las ventajas de ASP es la de que al utilizar VBScript, hace relativamente fácil escoger un lenguaje si ya se conoce como programar en Visual Basic. El soporte en ASP está habilitado por defecto en el servidor IIS haciendo fácil ponerlo y correrlo. Los componentes construidos en ASP están realmente limitados, de forma que si es necesario usar opciones "avanzadas" como interacción con servidores de FTP, usted necesitará comprar componentes adicionales. Existen un convertidores de ASP a PHP, asp2php es el más comúnmente usado.

1.2.5.2 PHP vs. Cold Fusion?

PHP se dice comúnmente que es más rápido y mas eficiente para tareas de programación compleja y para llevar a cabo nuevas ideas. PHP es referido comúnmente como más estable y menor consumidor de recursos. Cold Fusion tiene un menor manejador de errores, abstracción de base de datos y manejo de fechas.

Un punto a favor de Cold Fusion es su motor de búsquedas que es excelente, pero un motor de búsqueda no es algo que deba ser incluido en un lenguaje de scripts para WEB. PHP se ejecuta en casi todas la plataformas; Cold Fusion solamente está disponible sobre Win32, Solaris, Linux y HP/UX, Cold Fusion tiene un buen IDE y es generalmente más fácil de comenzar a usarlo, en cambio PHP inicialmente requiere más conocimientos en programación. Cold Fusion no está diseñado teniendo a programadores en mente, mientras PHP está orientado a programadores.

1.2.5.3 PHP vs. Perl?

La más grande ventaja de PHP sobre PERL es que PHP ha sido diseñado para realizar scripts para el WEB, mientras que PERL fue diseñado para hacer muchas más cosas y para obtener estas cosas puede resultar muy complicado. PHP tiene un formato menos confuso y más estricto sin pérdida de flexibilidad. PHP es más fácil de integrar dentro de HTML existente que PERL. PHP tiene muchas de las buenas funcionalidad de PERL: constructores, sintaxis, etc., sin hacerlo tan complicado como PERL. PERL es un verdadero lenguaje, ha existido desde el final de la década de 1980, y PHP está madurando muy rápidamente.

1.2.6 Comparación cualitativa entre CGIs, Servlets, JSP, y ASP.

A continuación se muestra una tabla con la comparación cualitativa entre las diferentes características de estas tecnologías:

Característica	CGI	Servlets	JSP	ASP	PHP
Aprendizaje	Mediano-Difícil	Mediano-Difícil	Mediano	Mediano	Mediano-Difícil
Codificación	Mediano-Difícil	Mediano-Difícil	Fácil	Fácil	Mediano-Difícil
Mantenimiento	Mediano-Difícil	Mediano-Difícil	Mediano	Mediano	Mediano
Independiente del servidor Web	Si	Sí	Sí	No	No
Independiente de la plataforma	Si	Sí	Sí	No	Sí

Tabla 1.1 Comparación cualitativa entre tecnologías.

- **Aprendizaje** – Nivel requerido por la herramienta para su aprendizaje con respecto a un programador principiante.
- **Codificación** – Nivel requerido por la herramienta para su codificación. Esto puede ser aplicado en general para cualquier nivel de programador.
- **Mantenimiento** – Nivel requerido para el mantenimiento correctivo y preventivo del código de las aplicaciones creadas con la herramienta. Esto puede ser aplicado en general para cualquier nivel de programador.

- **Independiente del Servidor Web** – Responde a la pregunta: Las aplicaciones creadas con la herramienta son independientes del servidor Web que se elija para su hospedaje y ejecución.
- **Independiente de la Plataforma** – Responde a la pregunta: Las aplicaciones creadas con las herramientas pueden ser ejecutadas no importando la plataforma que se desee utilizar.

1.3 Recursos específicos

En la actualidad existen una gran variedad de recursos disponibles para el usuario brindados por diferentes instituciones o empresas a los cuales se puede acceder remotamente y obtener un servicio más versátil y no limitarse solamente a su uso en forma local. Dentro de los recursos que más auge tienen en la actualidad tenemos: **comercio electrónico, bases de datos, bibliotecas digitales, laboratorios virtuales** y tipos de recursos específicos como es el caso del **laboratorio de paralelismo** de la Sección de Computación. Aunque en principio, el diseño de aplicaciones sobre Internet para cada recurso específico se utiliza el modelo C/S, cada uno de ellos tiene aspectos específicos que determinan las características de sus aplicaciones. A continuación se revisa rápidamente las características de los recursos antes mencionados.

1.3.1 Bases de datos y ODBC

Una base de datos es una estructura de datos homogéneos organizados de tal manera que se minimizan los efectos asociados a su manejo. Para que dicho manejo resulte fácil a los usuarios y que éstos dispongan de herramientas que faciliten la gestión completa, aparecen los sistemas gestores de bases de datos. Éstos son sistemas que *envuelven* y *protegen* los datos, en la medida de lo posible, frente a manipulaciones indebidas, al mismo tiempo que integran una serie de herramientas que gestionan, entre otras cosas, la manipulación completa de los datos, los accesos concurrentes, las copias de seguridad, la *integración* con programas en lenguajes de uso general, etc. Pero, sobre todo, los gestores proporcionan a los datos una característica básica, que es la independencia respecto de los programas que los usan y que lleva a evitar la redundancia no deseada.

Estos sistemas que se acaban de describir, y que se han denominado **gestores de bases de datos**, pueden representar la información de diferentes maneras (modelo jerárquico, en red, etc.), aunque la más utilizada en la creación de aplicaciones Web para el acceso a bases de datos es la proporcionada por el denominado *modelo relacional*. Este modelo se basa en el concepto de *relación* y la representación que hace es la correspondiente a la utilización de *tablas* o *relaciones* para el almacenamiento de datos, de tal forma que cada fila de la tabla es uno de los elementos que se desean guardar, y cada una de las columnas es una propiedad de dichos elementos. Si se hiciese una correspondencia con el concepto tradicional de *fichero*, se podría decir que una fila se corresponde con un registro y que una columna lo hace con un campo de dicho registro. A cada una de estar columnas se le denomina, en este entorno, *atributos*, y a cada fila, *tupla*.

Debido a que existe una gran variedad de gestores de bases de datos, existe la necesidad de contar con un mecanismo que permita acceder a cada uno de ellos de una manera estándar, ODBC (**O**pen **D**ata**B**ase **C**onnectivity) ofrece un acceso estándar a bases de datos. El objetivo de ODBC es el de hacer posible el acceso a cualquier dato desde cualquier aplicación, sin importar cual gestor de bases (DBMS por sus siglas en Inglés) de datos administre la información.

ODBC implementa una capa media llamada *driver* entre el gestor de bases de datos utilizado y la aplicación). El propósito de esta capa es la de traducir las *peticiones (queries)* de información de la aplicación en comandos entendibles por el gestor de bases de datos[24].

1.3.2 Comercio Electrónico

Cerca de 100 millones de usuarios y al menos un millón de negocios son actualmente accedidos, esto ha sido posible gracias a Internet. Esta masa crítica es comparable a la economía de mercado de Norte América, Japón o Europa. Pero el hacer negocios electrónicamente significa movilizar procesos críticos de negocios a redes de trabajo abiertas. También significa conectar aplicaciones de negocios de una manera flexible y segura.

1.3.2.1 Fases del Comercio Electrónico

En contraste con el modelo convencional de mercadeo, los sistemas de mercadeo electrónico básicamente hacen que las distancias geográficas sean no-relativas, permiten que la selección de productos se automatice y minimiza los costos totales de transacción. Una transacción comercial puede ser evaluada en términos de dos fases: *diseño* y *ejecución*. Sin embargo, cada fase contiene un número de subconjunto de fases.

Diseño. La fase de diseño comprende la construcción de un servicio comercial mediante un proveedor o consumidor.

1) Obtención de información – involucra el uso de catálogos y un sistema para buscar y localizar agencias en-línea, proveedores de servicio y consumidores.

2) Contrato – involucra el diseño dinámico de perfiles descriptivos para la creación de contratos de aplicación. (Un contrato son los términos y condiciones de un acuerdo realizado por las partes involucradas mediante negociaciones y otros mecanismos.)

3) Acuerdo – involucra la transición contrato-consolidación, caracterizada por firmar un documento en presencia de una entidad legal representativa.

Ejecución. Esta fase consiste en la manera en como la transacción se lleva a cabo.

1) Configuración – involucra el despliegue dinámico de una política de macro-información entre un grupo de participantes involucrados en una transacción de comercio electrónico (i.e. una política común de seguridad).

2) Ejecución de servicio – involucra la ejecución de servicios comerciales en el contexto de una política de contrato de alto-nivel.

3) Terminación – involucra la validación y el cierre del contrato entre todos los participantes (de acuerdo a las políticas).

La fase de diseño puede acelerarse mediante la estandarización de protocolos de negociación, servicios de pago electrónico y herramientas genéricas de acceso a servicios, hoy en día, Internet solo provee un soporte parcial para realizar esto.

La economía de la información aún carece de una infraestructura de software que permita a todos los participantes realizar transacciones de negocios. Sin embargo esto puede mejorarse mediante una infraestructura de comercio electrónico adecuada [8] [27].

1.3.3 Bibliotecas Digitales

Una biblioteca digital es una colección de información que se guarda y accede electrónicamente. Toda la información contenida en la Biblioteca debe referirse a un tema común. Es decir, una biblioteca digital puede diseñarse para gráficas por computador, sistemas operativos, redes o cualquier otro tema. Todas estas librerías separadas pueden combinarse bajo una misma interfaz, pero es esencial que la información contenida dentro de cada una se mantenga separada.

El propósito de estas bibliotecas es mantener una locación central para el acceso a información sobre un tema en particular. Una biblioteca digital debe guardar los temas separados, de otra manera sería totalmente inútil. También es de gran importancia el hecho de tener una interfaz al usuario que sea fácil de usar.

La construcción de una biblioteca digital es costosa y requiera una gran cantidad de recursos. Antes de embarcarse en tal aventura, es importante considerar algunos principios básicos subyacentes al diseño, implementación, y mantenimiento de una biblioteca digital.

Estos principios aplican no solamente para proyectos de conversión en el cual objetos análogos son convertidos a forma digital, sino también para bibliotecas digitales en la cual los objetos siempre han tenido forma digital y para "mezclar" bibliotecas digitales en las cuales los objetos pueden ser de ambos tipos. Los principios son, en cierto sentido, evidentes por sí mismos.

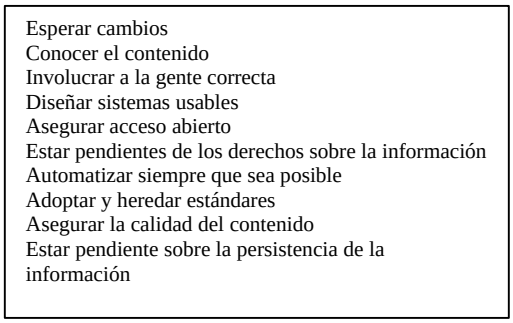


Figura 1.8 Principios para la construcción de bibliotecas digitales.

Los diez principios mostrados en la Figura 1.8 son derivados de la experiencia de varios autores en el desarrollo de sistemas de bibliotecas digitales durante la década pasada[3]. Estos principios se describen brevemente a continuación [21].

Esperar cambios. Las tecnologías cambiantes pueden rápidamente sobrepasar la habilidad de los diseñadores para mantener una biblioteca digital en particular. Un procedimiento que anticipe y planee los cambios es necesario para proveer un acceso duradero de la información de la biblioteca.

Conocer el contenido. Para los usuarios, el contenido es el aspecto más interesante y valioso de una biblioteca digital. Los creadores de bibliotecas digitales necesitan manejar y tomar decisiones concisas acerca del contenido de la biblioteca.

Involucrar a la gente correcta. Los dos aspectos más directamente involucrados en una biblioteca digital son la ciencia computacional y la ciencia de la biblioteca. Los científicos computacionales aprecian las posibilidades, así como las limitaciones de la tecnología y son generalmente los que construyen el sistema.

[3] McCray, A., Gallagher, M., y Flannick, M., **Extending the role of metadata in a digital library sistem**, IEEE Computer Society

Los bibliotecarios, incluyendo catalogadores y archivistas, han sido los custodios de las fuentes de información, comprendiendo no solamente las necesidades de información de diversas audiencias sino los pormenores envueltos en la preservación de materiales para un uso y acceso continuos. La búsqueda y desarrollo de las bibliotecas digitales ha significado que cada grupo comprenda las perspectivas del otro.

Diseñar sistemas usables. La mayoría de las bibliotecas digitales están diseñadas para ser utilizadas a través de Internet utilizando la tecnología Web, aunque, estrictamente hablando, esto no es un atributo necesario. Sin embargo, como las ventajas de la Web son muy buenas, la mayoría de los sistemas de biblioteca de hoy son diseñados para ser accedidos por Internet.

Asegurar acceso abierto. Asegurar el acceso abierto esta fuertemente relacionado con la usabilidad, incluyendo el acceso a la información en la biblioteca digital, así como a la misma biblioteca. Una manera de asegura el acceso abierto al contenido es evitando soluciones de hardware y software propietarios lo más posible.

Estar pendientes de los derechos sobre la información. Un posible riesgo al acceso abierto a la información surge debido a los derechos intelectuales. Las leyes de derecho de autor existentes proveen protección legal y económica a los publicistas de artefactos físicos. No existen respuestas directas para la aplicación de las leyes de derecho de autor disponibles en forma digital. El Internet y la Web emergieron de comunidades que creen en la libertad de compartir información, en lugar de restringirla.

Automatizar siempre que sea posible. Debido a que la construcción de una biblioteca digital requiere un gran esfuerzo intelectual por parte de los creadores del sistema, entre más herramientas automatizadas puedan crearse y usarse, mejor será el uso de los recursos humanos.

Adoptar y heredar estándares. El uso de estándares en la construcción del sistema tiene muchos beneficios. Las aplicaciones son más escalables, interoperables, y portables; estas características son importantes para el diseño, implementación, y mantenimiento de bibliotecas digitales.

Asegurar la calidad del contenido. Métricas de calidad pues aplicarse a todos los procesos y resultados involucrados en la creación de bibliotecas digitales.

Estar pendiente sobre la persistencia de la información. Múltiples sugerencias han sido hechas por investigadores para la preservación de objetos digitales. Quizá el punto más comúnmente discutido es el de la “estrategia de migración”, la cual exige la transformación de datos de un formato a otro, la conversión de un ambiente de software a otro, o la mudanza de un medio físico a otro.

1.3.4 Laboratorios Virtuales

Durante los años recientes, el interés por las aplicaciones didácticas tanto como herramientas de software para el desarrollo de instrumentos virtuales y herramientas para la interconexión de computadoras a gran distancia se ha incrementado constantemente. Este interés se debe principalmente al costo de los laboratorios experimentales en universidades con un gran número de estudiantes. En segundo lugar, pero no menos importante, es la importancia de enseñar métodos, basados en el uso correcto de nuevas tecnologías. Además, el aprendizaje a distancia para un entrenamiento continuo se está convirtiendo en un factor clave para mantener el liderato e improvisar la calidad de producción, productos y personal en muchas pequeñas y medianas empresas.

Desde este punto de vista, no se espera que los simuladores de instrumentos reemplacen los instrumentos reales. Los instrumentos reales y las prácticas de campo no pueden ser reemplazados en planes de estudio de temas experimentales, como las mediciones eléctricas y electrónicas y la instrumentación. Sin embargo, puede ser una extraordinaria herramienta didáctica auxiliar para el estudiante, para ayudarle a familiarizarse con los instrumentos y sus controles y operaciones de una manera remota [11].

Sin embargo, la actual efectividad de estos nuevos métodos didácticos está condicionada por la solución de los siguientes problemas prácticos:

- *La construcción de simuladores es un trabajo que consume mucho tiempo*. Las herramientas de desarrollo de software basadas en métodos de programación gráficos y orientados a objetos, hacen esta tarea más fácil, pero generalmente son costosas y se deben comprar licencias de tiempo-de-ejecución para correr el simulador, y no solamente la herramienta de trabajo. Algunas medidas de precaución deben tomarse cuan el acceso al simulador es permitido fuera de las instalaciones.
- *La más alta efectividad de los laboratorios didácticos virtuales se consigue cuando los laboratorios pueden ser accedidos remotamente.* De esta manera se les brinda a los estudiantes la posibilidad de revisar las propiedades de los instrumentos y sus características en cualquier momento y lugar.
- *El medio más adecuado actualmente para el acceso remoto es Internet*. Los simuladores pueden ser instalados en un servidor Web, y el acceso a estos recursos pueden ser organizados como páginas Web. Si esta solución es adoptada, el software residente debe protegerse adecuadamente, y solamente algunos archivos deben ser instalados en la máquina del cliente, ya que el tiempo de acceso al servidor es limitado.
- *El compartir recursos e información entre diversas instituciones de educación e investigación*. El uso de una red global de comunicación como Internet, y lenguajes de alto nivel y altamente estandarizados como HTML y Java permiten la colocación de diferentes programas de simulación en diferentes servidores, posiblemente localizados a larga distancia unos de otros. De esta manera diferentes instituciones de educación e investigación con planes de estudio similares pueden unirse en un grupo de desarrollo y distribuir la carga de desarrollo de los simuladores entre un gran número de programadores. Cada institución deberá desarrollar un simulador, ponerlo en su propio servidor Web, y hacerlo accesible a los demás miembros del equipo.

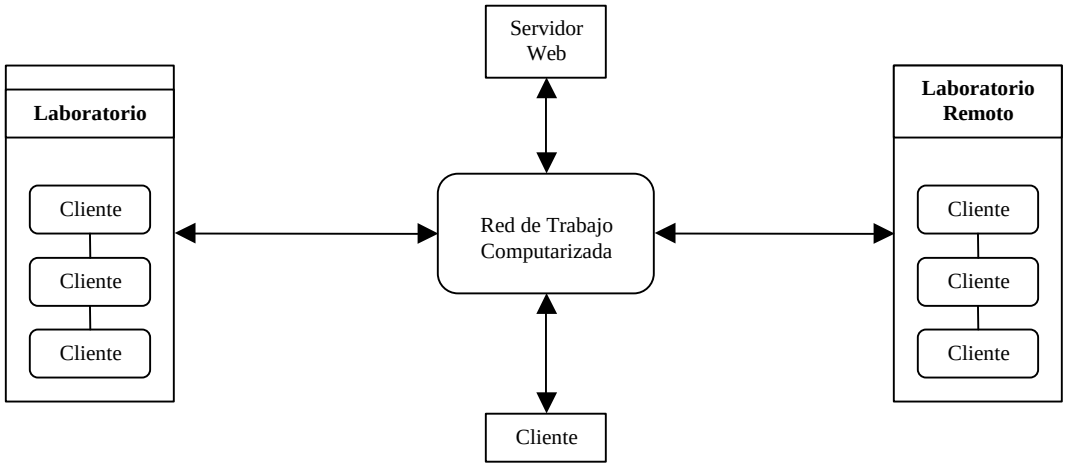


Figura 1.9 Ambiente distribuido cliente servidor.

1.3.5 Laboratorio de Paralelismo

Este tipo de recurso cuenta con un gran auge en la actualidad debido en gran parte a la tecnología Beowulf para cluster de estaciones de trabajo. Un cluster Beowulf es una computadora paralela de alto desempeño construida a partir de varias computadoras interconectadas por una red de trabajo de alta velocidad. Es utilizada para ejecutar tareas de cómputo de alto desempeño.

El cluster consiste de un nodo servidor y varios nodos esclavos construidos a partir de hardware común (como el de una PC promedio) y software (que en la gran mayoría de las veces es libre). El usuario interactúa con el nodo servidor, el cual provee una interfaz similar a la de cualquier computadora de escritorio, excepto que está conectada a los nodos esclavos a través de una red de trabajo de alta velocidad. Los nodos esclavos consisten como mínimo de un CPU, memoria RAM y una tarjeta de red. Los nodos esclavos pueden inicializarse (booted) a través de la red de trabajo y podrían no requerir monitor; solo son utilizados para procesar información.

Un cluster Beowulf típicamente utiliza alguna distribución de Linux como sistema operativo y software de librerías de paso-de-mensajes como lo son MPI (Message Passing Interface) o PVM (Parallel Virtual Machine) las cuales permiten al usuario escribir programas paralelos de paso-de-mensajes. Los factores principales que limitan el desempeño de un cluster Beowulf son la velocidad de los CPUs y los retardos de comunicación en la red de trabajo. Dependiendo de la aplicación, cualquiera de estos factores puede representar un cuello de botella crítico [28][26].

Para una institución que cuente con este tipo de computadora paralela sería de gran utilidad que se pudiera utilizar de una manera remota para así poder ampliar su campo de uso y aplicación, eliminando la necesidad de tener que trasladarse hasta su ubicación física. Tal es el caso del Laboratorio de Paralelismo con el que cuenta la Sección de Computación.

Capítulo 2

Laboratorio de Paralelismo

Este capítulo presenta diversas arquitecturas paralelas las cuales pueden tomarse como base para la implementación de un laboratorio de paralelismo, las herramientas disponibles para la creación de programas paralelos utilizando el paradigma de paso de mensajes, y algunos resultados obtenidos de pruebas de comunicación realizadas en el laboratorio de paralelismo de la sección.

2.1 Arquitecturas Paralelas

Las arquitecturas de computadoras, la tecnología, y las aplicaciones han evolucionado conjuntamente y existe una estrecha relación entre ellas. La arquitectura de computadoras paralelas no es la excepción. Una nueva dimensión ha sido añadida al diseño de espacio – el número de procesadores – y el diseño en general está fuertemente manejado por la demanda de desempeño a un costo razonable. Cualquiera que sea el desempeño de un solo procesador en un tiempo dado, un mayor desempeño podrá obtenerse, en principio, mediante la utilización de muchos de estos procesadores.

Dado que una computadora paralela es "una colección de elementos de procesamiento que cooperan y se comunican para resolver grandes problemas rápidamente [6]", una arquitectura paralela puede verse como la extensión de una arquitectura de computadora convencional para tratar tópicos de comunicación y comunicación entre elementos de procesamiento. En esencia, las arquitecturas paralelas extienden los conceptos usuales de una arquitectura de computadora con una *arquitectura de comunicación*. Las arquitecturas de computadora tienen dos facetas distintas. Una es la definición de abstracciones críticas, especialmente los límites de hardware/software y usuario/sistema. La arquitectura especifica el conjunto de operaciones en el límite y los tipos de datos que con los cuales operarán. La otra faceta es la estructura organizacional que realiza estas abstracciones para brindar un alto desempeño de manera efectiva y poco costosa.

Una *arquitectura de comunicación* también cuenta con estas facetas. Define las operaciones básicas de comunicación y sincronización, y trata con las estructuras organizacionales que realizan estas operaciones.

El marco de trabajo para entender la comunicación en una máquina paralela se ilustra en la Figura 2.1. La capa superior corresponde al modelo de programación, el cual es la conceptualización de la máquina que el programador utiliza para codificar aplicaciones.

Cada modelo de programación especifica cómo las partes del programa ejecutándose en paralelo intercambian información unas con otras y qué operaciones de sincronización están disponibles para coordinar sus actividades. Las aplicaciones están escritas en un modelo de programación el cual, en el caso más simple, consiste en la multiprogramación de gran número de programas secuenciales independientes; no se llevan a cabo operaciones de comunicación o cooperación. Sin embargo, en el caso más interesante incluye verdaderos modelos de cooperación y sincronización. Estos modelos se pueden describir brevemente como sigue:

- La programación con *memoria compartida* es como utilizar un tablero de información, donde uno se puede comunicar con una o varias tareas poniendo información en una ubicación compartida y bien conocida. Las actividades individuales puede ser orquestadas tomando nota de quién esta haciendo qué tarea.
- El *intercambio de mensajes* es semejante a las llamadas telefónicas o cartas, donde se lleva información de un emisor a un receptor. Existe un evento bien definido cuando la información es enviada o recibida, y estos eventos son la base para orquestar las actividades individuales. Sin embargo, no existe en lo absoluto alguna locación compartida.
- Procesamiento de *datos paralelos* es una forma más estricta de cooperación, donde múltiples agentes desempeñan simultáneamente una acción en elementos separados de un conjunto de datos y después intercambian información globalmente antes de continuar.

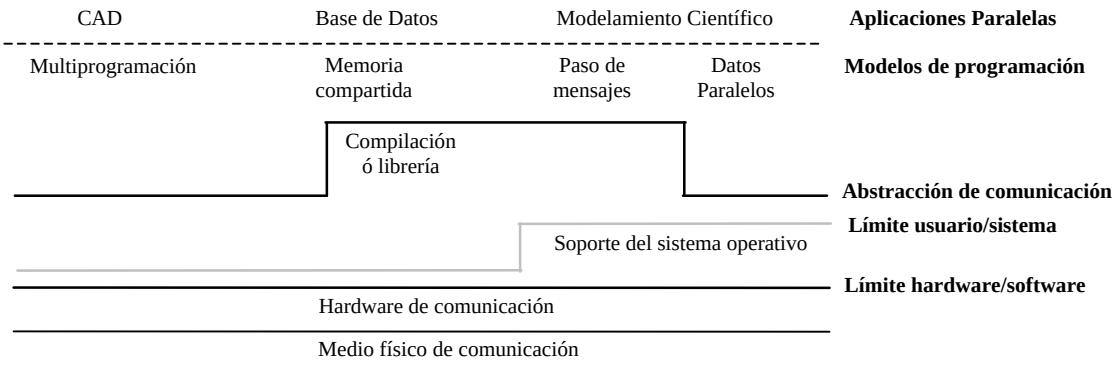


Figura 2.1 Capas de abstracción en un arquitectura paralela de computadora.

De acuerdo a la Figura 2.1 las capas críticas de abstracción recaen entre el programa de aplicación y el hardware actual. La aplicación es escrita para un modelo de programación, el cual dictamina como las piezas del programa compartirán información y coordinarán sus actividades. Las operaciones específicas proveen comunicación y sincronización conformando la abstracción de comunicación, la cual es el límite entre el programa de usuario y la implementación del sistema. Esta abstracción es realizada a través de soporte de compilador o librerías utilizando las primitivas disponibles del hardware o del sistema operativo. El hardware de comunicación esta organizado para proveer estas operaciones eficientemente en las conexiones físicas.

2.1.1 Espacio de Direccionamiento Compartido

Una de las clases más importantes de máquinas paralelas son los *multiprocesadores de memoria compartida*. La propiedad clave de esta clase es que la comunicación entre procesos ocurre implícitamente como resultado de instrucciones convencionales de acceso a memoria (como load y store).

El hardware de comunicación para multiprocesadores de memoria compartida es una extensión natural del sistema de memoria encontrado en la mayoría de las computadoras. Esencialmente todos los sistemas de cómputo permiten un procesador y un conjunto de controladores de Entrada/Salida (E/S) para acceder a una colección de módulos de memoria a través de algún tipo de hardware interconectado, como se ilustra en la Figura 2.2.La capacidad de memoria es incrementada simplemente añadiendo módulos de memoria.

La capacidad adicional puede o no puede incrementar el ancho de banda de memoria existente, dependiendo en la organización específica del sistema. La capacidad de E/S es incrementada añadiendo dispositivos a controladores de E/S o insertando controladores de E/S adicionales. Existen dos posibles maneras de incrementar la capacidad de procesamiento: esperar que este disponible un procesador más rápido o añadiendo más procesadores.

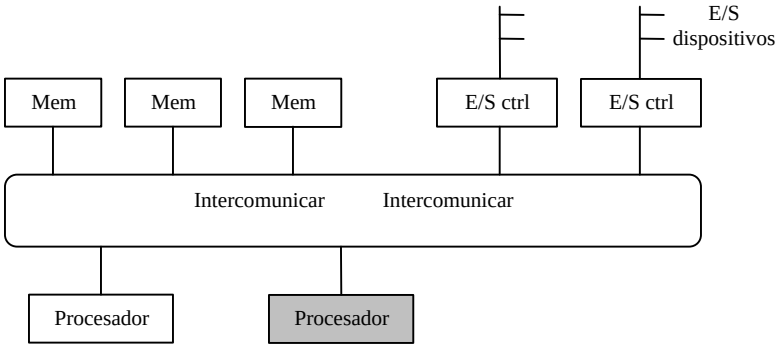
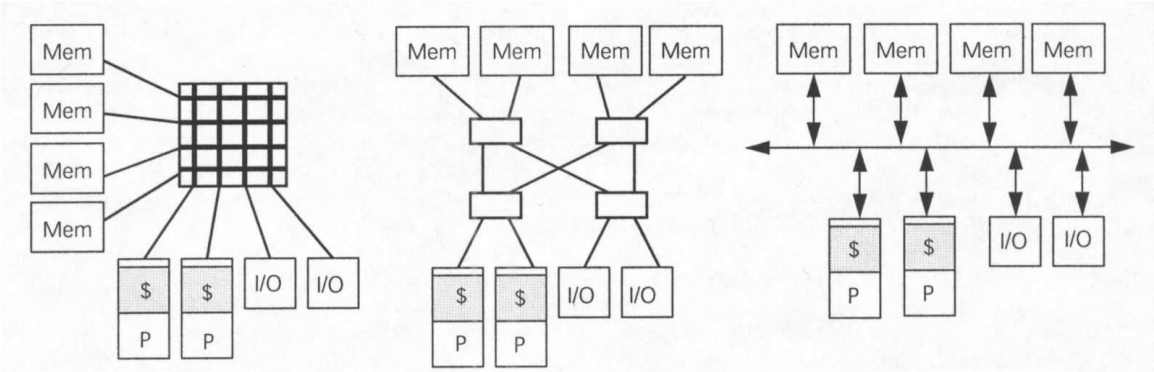


Figura 2.2 Extendiendo un sistema en uno multiprocesador de memoria compartida mediante la adición de módulos. La mayoría de los sistemas consisten de uno o más módulos de memoria accesibles por un procesador y controladores de E/S a través de hardware interconectado, típicamente un bus, barra de cruce (crossbar) ó interconectando multietapas (multistage). La capacidad de memoria y de E/S es incrementada adjuntando módulos de memoria y de E/S. Las máquinas de memoria compartida permiten que la capacidad de procesamiento sea incrementada mediante la adición de módulos de procesadores (mostrados en gris).



a) Switch de barra de cruce (Crossbar switch)

b) Red de trabajo de interconexión multietapas (Multistage interconnection network)

c) Interconexión en bus (Bus interconnect)

Figura 2.3 Esquemas típicos de interconexión de multiprocesadores de memoria compartida. La interconexión de múltiples procesadores, con sus caches locales (indicados por $), y controladores de E/S a múltiples módulos de memoria utilizando una barra de cruce, una red de trabajo de interconexión multietapas, ó un bus.

2.1.2 Memoria distribuida

Una segunda clase importante de las máquinas paralelas es conocida como *arquitecturas de memoria distribuida* o de *paso de mensajes*. Esta emplea computadoras enteras como bloques de construcción – incluyendo el microprocesador, memoria, y el sistema de E/S – y provee comunicación entre procesadores como operaciones explícitas de E/S.

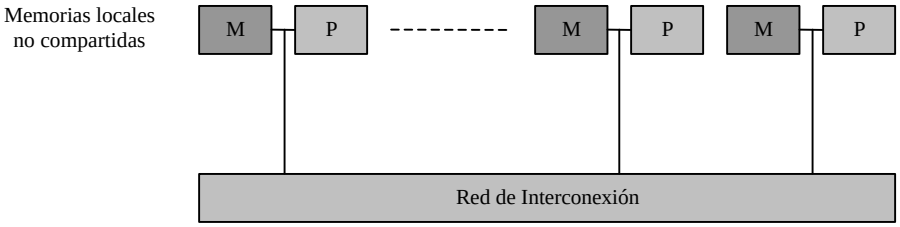


Figura 2.4 Arquitectura de paso de mensajes.

Las operaciones de comunicación a nivel de usuario más utilizadas en sistemas de paso de mensajes son variantes de *send* y *receive*. En su forma más simple, *send* especifica un buffer de datos locales que es transmitido y recibido por un proceso (típicamente en un procesador remoto). *Receive* especifica un proceso de envío y un buffer de datos locales en el cual los datos transmitidos serán almacenados. Juntos, los correspondientes *send* y *receive* causan una transferencia de datos de un procesador a otro, como se indica en la Figura 2.5.

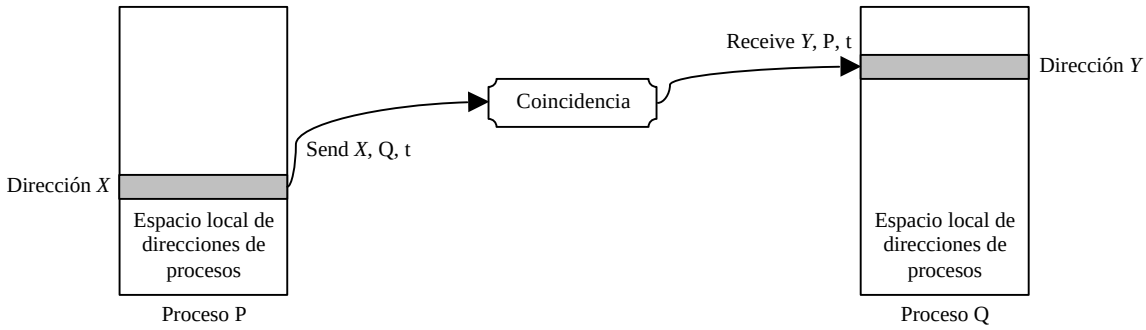


Figura 2.5 Abstracción del paso de mensajes a nivel de usuario send/receive. Una transferencia de datos de un espacio local de direcciones a otro ocurre cuando un *send* a un proceso en particular coincide con un *receive* enviado por ese proceso.

En la mayoría de los sistemas de paso de mensajes, la operación *send* también permite un identificador o *tag* añadido al mensaje, y la operación *receive* especifica una regla de coincidencia (como un tag específico proveniente de un procesador específico o cualquier tag proveniente de cualquier procesador). La combinación de un *send* y un *receive* coincidente realiza un evento de sincronización en pareja y una copia de memoria-a-memoria, donde cada lado especifica sus direcciones de datos locales.

Existen muchas variantes de este evento de sincronización, dependiendo de si la operación *send* se completa cuando la operación *receive* ha sido ejecutada, cuando el buffer de envío esta disponible para reutilizarse, o cuando la solicitud de envío ha sido aceptada. Similarmente, la operación de *receive* puede esperar hasta que ocurra una operación *send* coincidente o simplemente ejecutarse sin esperar por alguna coincidencia [6].

Además de las primitivas básicas, *send* y *receive*, para comunicación entre dos procesos, existen patrones de comunicación típicos que se presentan en aplicaciones para memoria distribuida. Estos patrones se identifican como primitivas para *comunicación colectiva* y *sincronización de procesos*, algunas de las más importantes describen brevemente a continuación (ver Figura 2.6):

- **Broadcast.** El tipo más simple de comunicación colectiva es el broadcast. En una operación broadcast un simple proceso envía una copia de un dato a todos los demás procesos en un grupo.
- **Gather y Scatter.** Las clases más importantes de las comunicaciones colectivas son aquellas que distribuyen los datos de un procesador a un grupo de procesadores o viceversa, esto es realizado por las operaciones scatter y gather, respectivamente.
- **Reduce.** Una reducción es una operación colectiva en la cual un solo proceso (el proceso raíz) colecta datos de los otros procesos presente en un grupo y los combina en una sola entidad de datos.

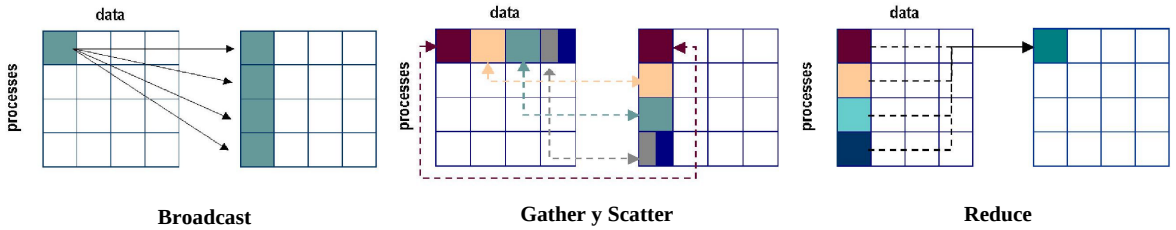


Figura 2.6 Primitivas de Comunicación Colectiva.

- **Barrier.** Una barrera se utiliza para realizar sincronizaciones colectivas de manera explícita. En un conjunto de procesos solo podrán continuar con su ejecución hasta que todos los procesos hayan alcanzado la barrera.

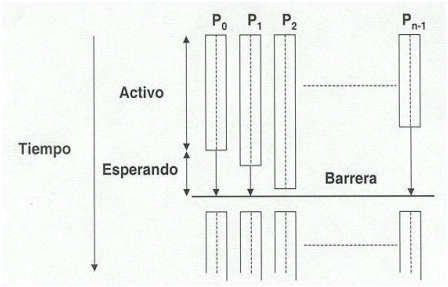


Figura 2.7 Primitiva Barrier de Comunicación Colectiva. Las regiones rectangulares representan tiempo de cómputo y las líneas punteadas tiempo ocioso (cuando no se usa el procesador) por el proceso P_i.

2.2 Programación Paralela

Desde el punto de vista más abstracto, un cómputo paralelo involucra la creación y ejecución de múltiples tareas ejecutándose en el mismo lapso de tiempo. Con base en lo anterior un modelo simple de programación paralela se muestra en la Figura 2.8.

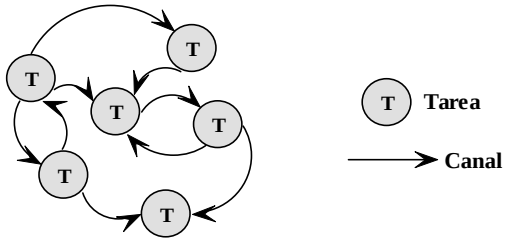


Figura 2.8 Un modelo simple de programación paralela (tarea/canal). Un cómputo consiste de un conjunto de tareas conectadas por canales. Una tarea encapsula un programa y una memoria local y define un conjunto de puertos que definen la interfaz de su ambiente. Un canal es una lista de mensajes en la cual un mensajero (sender) puede poner mensajes y de la cual un receptor (receiver) puede remover mensajes "bloqueándose" si no existen mensajes disponibles.

Este modelo puede resumirse como sigue:

1. Una tarea encapsula un programa secuencial y una memoria local. En adición, un conjunto de *puertos de entrada* y *puertos de salida* definen la interfaz de su ambiente.
2. Una tarea puede desempeñar cuatro acciones básicas en adición a las de lectura y escritura en su memoria local (ver Figura 2.9): enviar mensajes en sus puertos de salida, recibir mensajes en sus puertos de entrada, crear una nueva tarea, y terminar.
3. Una operación *send* es asíncrona: se completa inmediatamente. Una operación *receive* es síncrona: causa que la ejecución de una tarea se bloquee hasta que un mensaje este disponible.

4. Los pares de puertos de entrada/puertos de salida pueden ser conectados por listas de mensajes llamados *canales*. Los canales pueden ser creados y eliminados, y referencias a canales (puertos) pueden ser incluidos en los mensajes, de tal forma que la conectividad puede variar dinámicamente.
5. Las tareas pueden mapearse a procesadores físicos de varias formas; el mapeo utilizado no afecta la semántica de un programa. En particular, múltiples tareas pueden ser mapeadas a un solo procesador.

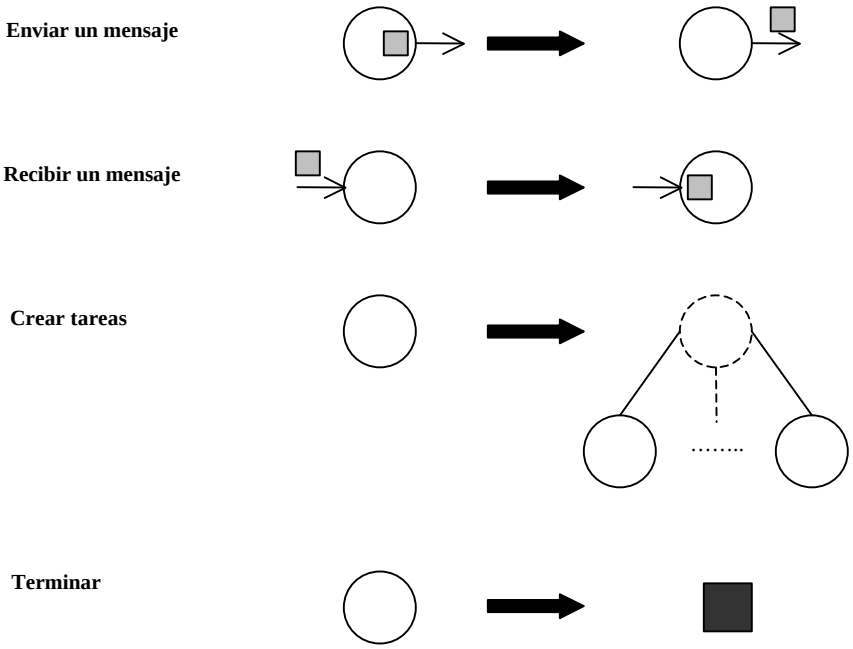


Figura 2.9 Las cuatro acciones básicas de una tarea. En adición a la lectura y escritura en memoria local, una tarea puede mandar un mensaje, recibir un mensaje, crear nuevas tareas, y terminar.

El modelo anterior permite representar cualquier tipo de procesamiento paralelo, sin embargo, desde el punto de vista práctico (de programación) y relacionado con el tipo de arquitectura de computadora con la cual se trabaja se han propuesto varios otros modelos. A continuación se describen brevemente tres de los modelos más importantes, su diferencia fundamental se encuentra en su flexibilidad, mecanismos de interacción de tareas, granularidad de tareas, soporte para locación, escalabilidad, y modularidad[12].

1. **Paso de Mensajes.** El paso de mensajes es probablemente el modelo de programación paralela más ampliamente utilizado hoy en día. Los programas de paso de mensajes, como los programas tarea/canal, crean múltiples tareas, cada una encapsulando sus datos de manera local. Cada tarea esta identificada por un nombre único, y las tareas interactúan enviando y recibiendo mensajes entre si. Con respecto a esto, el paso de mensajes es realmente una variación menor del modelo tarea/canal, difiriendo solamente en el mecanismo utilizado para la transferencia de datos.
2. **Paralelismo de Datos.** El paralelismo de datos hace un llamado a la explotación de simultaneidad que se deriva de la aplicación de una misma operación a múltiples elementos de una estructura de datos, por ejemplo, “añadir 2 a todos los elementos de este array”. Un programa de paralelismo de datos consiste de una secuencia de este tipo de operaciones. Cada operación en cada uno de los elementos puede pensarse como una tarea independiente, la granularidad natural de un cómputo de datos paralelos es pequeña, y el concepto de “localidad” no surge naturalmente. Así, los compiladores de datos paralelos comúnmente requieren que el programador provea la información acerca de cómo los datos serán distribuidos entre los procesadores, en otras palabras, como los datos serán particionados en tareas.
3. **Memoria Compartida.** En el modelo de programación de memoria compartida, las tareas comparten direcciones de memoria comunes, sobres las cuales pueden leer y escribir asíncronicamente. Varios mecanismos como bloqueos (locks) y semáforos pueden utilizarse para controlar el acceso a la memoria compartida.

2.2.1 Diseño de Algoritmos Paralelos

El primer paso en la construcción de un algoritmo paralelo destinado a la resolución de un problema específico es descomponer el problema en problemas más pequeños. Entonces dichos problemas son asignados a procesadores para trabajar simultáneamente. Estrictamente hablando, existen dos tipos de descomposición:

- **Descomposición de Dominio.** En la descomposición de dominio o paralelismo de datos, los datos son divididos en piezas de aproximadamente el mismo tamaño y mapeados a diferentes procesadores. Cada procesador entonces trabaja solamente en la porción de datos que se le asignó. Este tipo de descomposición del problema es ampliamente utilizado en las arquitecturas paralelas de memoria distribuida que utilizan librerías de paso de mensajes.
- **Descomposición Funcional.** En la descomposición funcional o paralelismo de tareas, el problema es descompuesto en un gran número de tareas pequeñas y entonces, las tareas son asignadas a los procesadores conforme se vuelven disponibles. El procesador que termine más rápidamente simplemente se le asigna más trabajo.

2.3 MPI

Una de las herramientas más utilizadas para la programación paralela con paso de mensajes es MPI. MPI (Message Passing Interface), es un estándar portable del paradigma de paso de mensajes que especifica la comunicación entre un conjunto de procesos. El estándar define la sintaxis y semántica de un conjunto de rutinas que se agrupan en una biblioteca de funciones. El objetivo de MPI es el de construir un estándar para la construcción de programas de paso de mensajes que sea ampliamente utilizado.

El estándar MPI fue desarrollado durante un periodo de 12 meses en 1993-1994 de juntas intensivas involucrando a más de 80 personas provenientes de aproximadamente 40 organizaciones, principalmente de U.S. y Europa. Este esfuerzo culminó en 1994 con la publicación de la especificación MPI[1]. MPI es utilizado para especificar la comunicación entre un conjunto de procesos que conforman un programa concurrente. El paradigma de paso de mensajes es atractivo debido a su gran portabilidad y escalabilidad. Es fácilmente compatible tanto con multicomputadoras de memoria distribuida como multiprocesadores de memoria compartida, con NOWs (Networks of Workstations), y con combinaciones de estos elementos. El paso de mensajes no será obsoleto si se incrementa la velocidad de la red de trabajo o por arquitecturas que combinen componentes de memoria compartida y distribuida.

El estándar especifica lo siguiente:

- *Comunicaciones punto-a-punto*. Mensajes entre un par de procesos.
- *Comunicaciones colectivas*. Operaciones de comunicación o sincronización que involucran a grupos enteros de procesos.
- *Grupos de procesos*. Como son manipulados y utilizados los grupos de procesos.
- *Comunicadores*. Un mecanismo que provee ámbitos de comunicación separados para módulos o librerías. Cada comunicador especifica un nombre de espacio distinto para procesos y un contexto de comunicación distinto para mensajes.
- *Topologías de Procesos*. Funciones que permiten una manipulación conveniente de etiquetas de procesos cuando los procesos se miran como formando un topología en particular, como lo sería un plano Cartesiano.
- *Vínculos para Fortran 77 y ANSI C*. MPI fue diseñado de tal manera que sus versiones de C y Fortran tuvieran una sintaxis directa. De hecho, la forma detallada de la interfaz en estos dos lenguajes esta especificado y es parte del estándar.

[1] (ftp://www.netlib.org/mpi/mpi-report.ps)

Varios tópicos importantes de la programación paralela no están cubiertos por el estándar, incluyendo:

- Operaciones de memoria compartida
- Mensajes manejadores de interrupciones, ejecución remota, y mensajes activos
- Herramientas de construcción de programas
- Soporte para depuración
- Soporte para hilos (threads)
- Manejo de tareas o procesos
- Funciones de entrada y salida

Las razones principales por las que estos tópicos no están abarcados se debió a que el comité se auto-impuso un periodo de tiempo para la finalización del estándar y porque observaron que varios de estos tópicos son dependientes del sistema en el que se manejen[9].

2.3.1 Implementaciones MPI

Existen una gran variedad de implementaciones funcionales de la especificación MPI que están disponibles para ambientes de cómputo distribuido y paralelos. Algunas de ellas son gratuitas y algunas son provistas por fabricantes de computadoras diseñadas específicamente para sus productos[2].

[2] (http://www.lam-mpi.org/mpi/implementations/fulllist.php)

Algunas de las distribuciones gratuitas se presentan en la Tabla 2.1.

Distribución	Desarrollada Por	Localización
CHIMP/MPI	Edinburg Parallel Computing Centre	http://www.epcc.ed.ac.uk/epcc-projects/CHIMP/index.html
CRI/EPCC MPI for Cray T3D	Cray Research Incorporated via EPCC	http://www.epcc.ed.ac.uk/t3dmpi/Product/
LAM/MPI	LAM Team	http://www.lam-mpi.org/
MPIAP	CAP Research Program, Australian National University	http://cap.anu.edu.au/cap/projects/mpi/mpi.html
MPI-FM	Concurrent Systems Architecture Group, University of Illinois	http://www-csag.ucsd.edu/projects/comm/mpi-fm.html
NT-MPICH	RWTH Scalable Computing Aachen	http://www.lfbs.rwth-aachen.de/~karsten/projects/nt-mpich/index.html
WMPI	Critical Software	http://www.criticalsoftware.com/wmpi
MacMPI	UCLA AppleSeed	http://exodus.physics.ucla.edu/appleseed/appleseed.html

Tabla 2.1 Distribuciones gratuitas del estándar MPI.

Algunas implementaciones provistas por vendedores de computadoras se presentan en la Tabla 2.2.

Distribución	Desarrollada Por	Ubicación
Alpha Data MPI	Alpha Data Parallel System Ltd.	http://www.alphadata.co.uk/softhome.html
HP-MPI	Hewlett-Packard, Co.; Convex Division	http://www.hp.com/rsn/mpi/mpihome.html
IBM Parallel Environment for AIX - MPI Library	IBM Corporation	http://www.qpsf.edu.au/software/ppe.html
MPI	Sillicon Graphics Inc	http://www.sgi.com/software/mpt/overview.html
MPI/SX	Nec Corporation	http://www.ccrl-nece.de/~mpi
T.MPI	Telmat Multinode	http://www.shu.ac.uk/schools/cms/student/amoore/openside/company/telmat.html

Tabla 2.2 Implementaciones del estándar MPI provistas por vendedores de computadoras.

2.3.2 MPICH

MPICH[3] es la primera implementación portable completa de MPI para una amplia variedad de plataformas que soportan el modelo de paso de mensajes, como lo son las supercomputadoras de memoria distribuida (MPPs), los multiprocesadores de memoria compartida (SMPs), y las redes de trabajo de estaciones de trabajo (NOWs). MPICH es un proyecto realizado en conjunto por los Laboratorios Nacionales Argonne (Hill Gropp y Rusty Luks) y la Universidad del Estado de Mississipi (Tony Skjellum y Nathan Doss).

[3] (http://www-unix.mcs.anl.gov/mpi/mpich/)

La reusabilidad de MPICH esta garantizada por la Interfaz de Dispositivo Abstracta (Abstract Device Interface - ADI), la cual provee una interfaz pequeña y eficiente para hardware y software específico de un fabricante en particular. La portabilidad de MPICH significa que puede ser utilizado en una gran variedad de ambientes paralelos, incluyendo computadoras paralelas y clusters de estaciones de trabajo.

MPICH soporta las siguientes arquitecturas: IBM SPx, Intel Paragon, Cray 3TD, Cray YMP, Cray PVP, Cray C90, Meiko CS2, TMC CM5, NCUBE NCUBE-2, Convex Exemplar, SGI (Power) Challenge, Sun Multiprocessors, 486's-Linux, Sun, HP, SGI, IBM (RS/6000) y DEC (Alpha).

Las características principales de **mpich** son:

- Acata completamente el estándar MPI 1.2, incluyendo cancelación de envíos (sends).
- Incluye compatibilidad para el estándar MPI-2 C++
- Incluye compatibilidad par Fortran 77 y Fortran 90
- Una versión para Windows NT está disponible.
- Soporta una gran variedad de ambientes, incluyendo clusters de SMPs y computadoras paralelas masivas.
- Mpich también incluye componentes de un ambiente de programación paralelo, incluyendo
    - Herramientas de visualización de desempeño paralelo (*upshot* y *jumpshot*)
    - Pruebas extensivas de corrección y desempeño
    - Herramientas de rastreo y seguimiento de eventos (tracing y logfile)

2.3.3 Aplicaciones MPI

La estructura típica de una aplicación escrita para MPI se presenta en la Figura 2.10. En esta estructura se distinguen tres partes importantes:

- **A.** Inicialización del sistema de comunicaciones (línea 9), obtención del número de procesadores a utilizar (línea 10) y asignación de un identificador a cada uno de los procesadores (línea 11).
- **B.** Procesamiento de datos dependiendo del identificador de cada procesador, en donde se pueden ver involucradas comunicaciones punto a punto o colectivas.
- **C.** Finalización del sistema de comunicaciones.

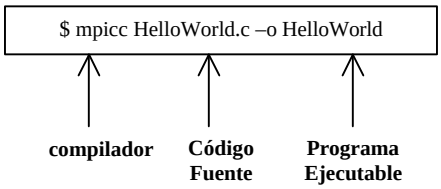
HelloWorld.c

```
1: #include “mpi.h”

2: int main(int argc, char* argv[])
3: {
4:          int myrank;          /* Rango del proceso */
5:          int tag = 1;         /* Etiqueta para los mensajes */
6:          int numprocs;        /* Número de procesadores */
7:          char message[20];    /* Almacenamiento para el mensaje */
8:          MPI_Status status;   /* Regresa el estatus para el receptor */

/*************************************************************/
9:          MPI_Init(&argc, &argv);
10:         MPI_Comm_size(MPI_COMM_WORLD, &numprocs);          A
11:         MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
/*************************************************************/

/*****************************************************************************************/
12:         if (myrank == 0) {
13:                   sprintf(message, “Hello World!”);
14:                   MPI_Send(message, 20, MPI_CHAR, 1, tag, MPI_COMM_WORLD);
15:                   printf(“Soy la máquina A. Estoy enviando un mensaje a la máquina B.\n”);
16:         }                                                                          B

17:         else (myrank == 1) {
18:                   MPI_Recv(message, 20, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status);
19:                   printf(“Soy la máquina B. El mensaje recibido es ‘%s’ \n”, message);
20:         }
/*****************************************************************************************/

/************************/
21:         MPI_Finalize();          C
/************************/

22: }
```

Figura 2.10 Código fuente de una aplicación MPI.

El código fuente de la Figura 2.10 corresponde a una aplicación MPI muy sencilla que utiliza las primitivas MPI_Send y MPI_Recv para intercambiar un mensaje entre 2 procesadores. En MPICH la forma para ejecutar una aplicación involucra:

1. **Compilación y generación del código ejecutable.** La implementación **mpich** provee cuatro comandos para compilar y ligar (link) programas en C (mpicc), C++ (mpiCC), Fortran 77 (mpif77), y Fortran 90 (mpif90). Para el código fuente de la Figura 2.10 se utiliza:



2. **Ejecución de la aplicación.** Para ejecutar un programa MPI, se utiliza el comando **mpirun**. Por ejemplo, para el código ejecutable obtenido de la compilación del código fuente de la Figura 2.10 (paso anterior), se utiliza:

```
$ mpirun –np 2 HelloWorld
```

para ejecutar el programa en 2 procesadores. Los procesadores asignados a la ejecución del programa los determina el comando **mpirun** en base a un archivo por defecto que contiene los nombres de las máquinas de procesamiento disponibles y utiliza un método Round-Robin. Este archivo tiene el nombre de *machines.XXX* dependiendo de la arquitectura utilizada (por ejemplo *machines.LINUX*) y su ubicación es *$MPI_HOME/share/* (dependiendo de la trayectoria elegida al momento de instalar la biblioteca). Este archivo sigue el formato indicado en la Tabla 2.3.

Cluster de Computadoras (Beowulf)	Máquinas Multiprocesadores
Máquina1	Máquina1:np
Máquina2	Máquina2:np
Máquina3	Máquina3:np
.	.
.	.
Máquina*n*	Máquina*n*:np

Tabla 2.3 Formato del archivo machines.XXX.

Dependiendo del tipo de arquitectura paralela utilizada, se le da el formato adecuado al archivo machines.XXX (ver Tabla 2.3). Para el caso de un cluster de computadoras, cada una de las líneas del archivo contiene el nombre de las máquinas que lo conforman. En el caso de que se utilicen máquinas multiprocesadores, cada una de las líneas contendrá el nombre de las máquinas disponibles además del número de procesadores con los que cuentan cada una de ellas. También se puede utilizar una combinación de ambos.

2.4 Laboratorio de Paralelismo

El Laboratorio de Paralelismo de la Sección de Computación esta conformado por dos servidores multiprocesadores interconectadas entre sí a través de una red FastEthernet, cuentan con el software necesario para la creación y ejecución de programas paralelos que utilizan el paradigma de paso de mensajes, además de compartir directorios utilizando NFS (Network File System) (ver Figura 2.11).

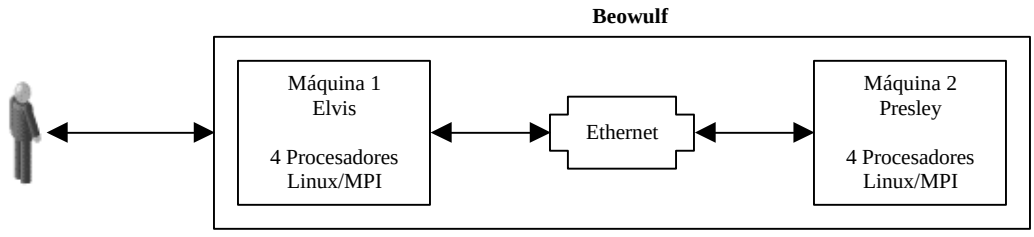


Figura 2.11 Esquema general del Laboratorio de Paralelismo.

En la siguiente tabla se describe con más detalle las características de cada una de estas computadoras:

	Máquina 1	**Máquina 2**
Nombre	elvis.cs.cinvestav.mx	presley.cs.cinvestav.mx
Número de Procesadores	4	4
Tipo de Procesador	Pentium III/750Mhz	Pentium III/550Mhz
Memoria RAM	1GB	1GB
Capacidad en Disco Duro	27GB	27GB
Tarjeta de Red	Intel Pro 10/100	Intel Pro 10/100
Sistema Operativo	Red Hat Linux 7.1	Red Hat Linux 7.1
Librería de Paso de Mensajes	MPI LAM/MPICH	MPI LAM/MPICH

Tabla 2.4 Características de las máquinas que conforman el Laboratorio de Paralelismo.

Como se puede notar, tal instalación cumple con los requisitos mínimos necesarios para la creación de una arquitectura Beowulf [28] [26].

2.4.1 Pruebas de Rendimiento

Una de las primeras actividades a realizar con un cluster de computadoras es determinar sus parámetros básicos del sistema de comunicaciones. Así, para obtener datos de referencia acerca del desempeño del las máquinas Elvis-Presley (ver Tabla 2.4), se realizaron varias pruebas de rendimiento:

- **Prueba de Comunicación Punto a Punto: Un Par.** En esta prueba solamente se involucran a un par de procesadores para el intercambio de mensajes utilizando las primitivas de comunicación *MPI_Send()* y *MPI_Recv()*. Solamente uno de los procesadores envía mensajes de longitud creciente y el otro los recibe. Esta prueba nos permite obtener la *latencia* y *ancho de banda pico* del sistema.
    - **Latencia.** Tiempo de arranque de las comunicaciones.
    - **Ancho de Banda Pico.** Velocidad para transferir una unidad de datos.

Debido a que las máquinas evaluadas cuentan con 4 procesadores cada una, esta prueba se realizó tanto para un par de procesadores locales (máquina Presley) como para un par de procesadores localizados en máquinas distintas (Elvis-Presley).

- **Prueba de Comunicación Punto a Punto: Pares Simultáneos.** En esta prueba se involucran a varios pares de procesadores para el intercambio de mensajes utilizando las primitivas de comunicación *MPI_Send()* y *MPI_Recv()*. Todos los procesadores involucrados envían y reciben mensajes entre sí. Esto permite obtener medidas de la contención y congestión que se presentan en la arquitectura de red utilizada en el sistema a prueba, también nos permite obtener el *ancho de banda* real.

En las Figuras 2.12 a 2.15 se muestran los resultados de la prueba de comunicación punto a punto entre un par de procesadores localizados en la misma máquina (Presley). Así, lo que se mide es el efecto de usar una computadora de memoria compartida mediante el paradigma de paso de mensajes.

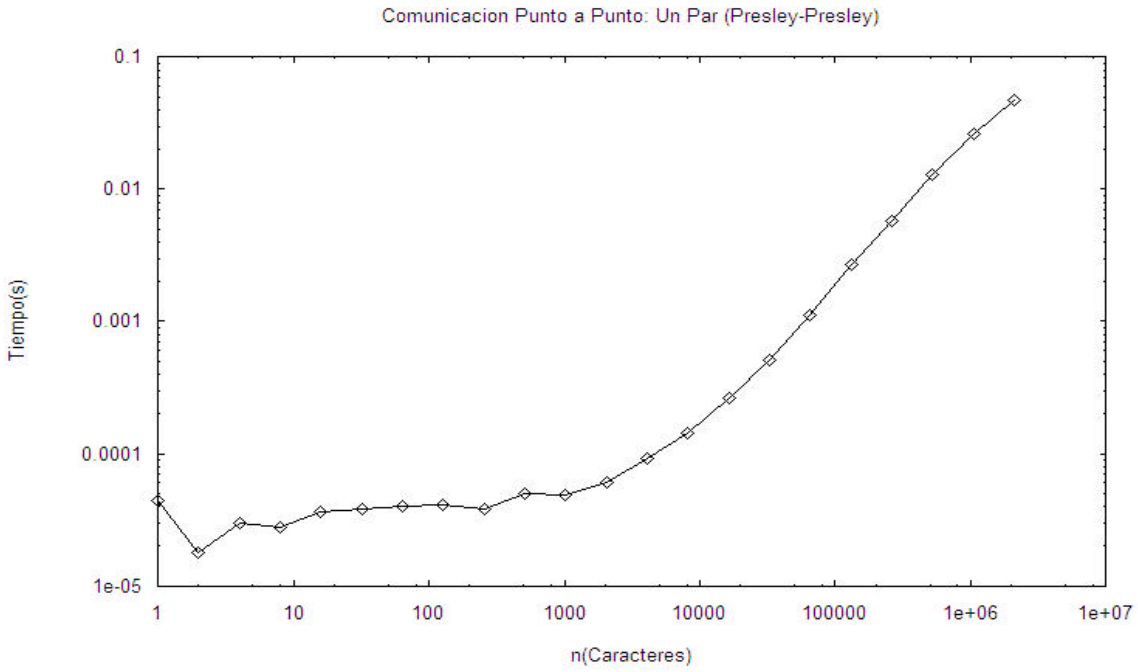


Figura 2.12 Prueba de Comunicación Punto a Punto: Un Par (Presley-Presley).

Para poder apreciar de una mejor manera los resultados de la Figura 2.12, se dividió en dos partes como se muestra en las Figuras 2.13 y 2.14:

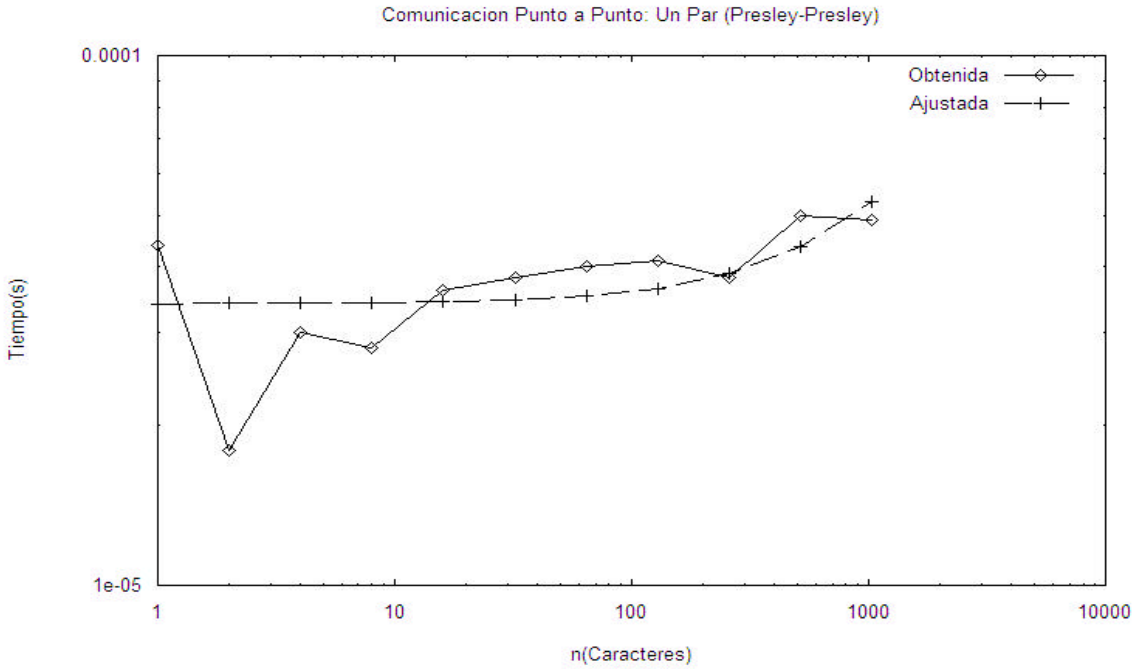


Figura 2.13 Prueba de Comunicación Punto a Punto: Un Par (Presley-Presley).

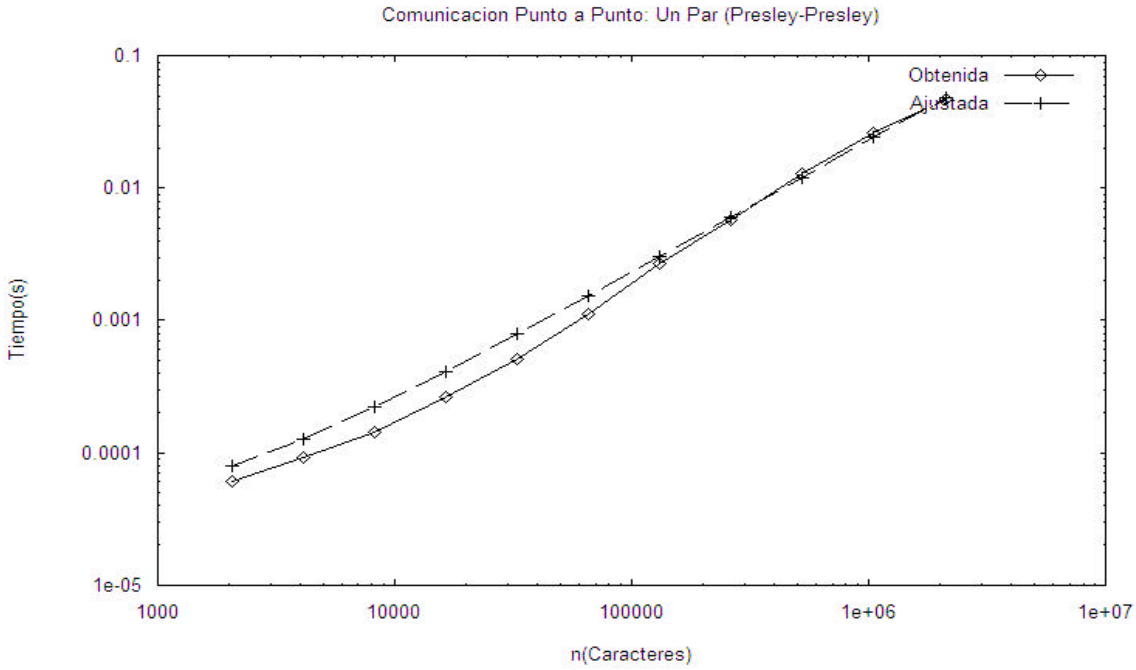


Figura 2.14 Prueba de Comunicación Punto a Punto: Un Par (Presley-Presley).

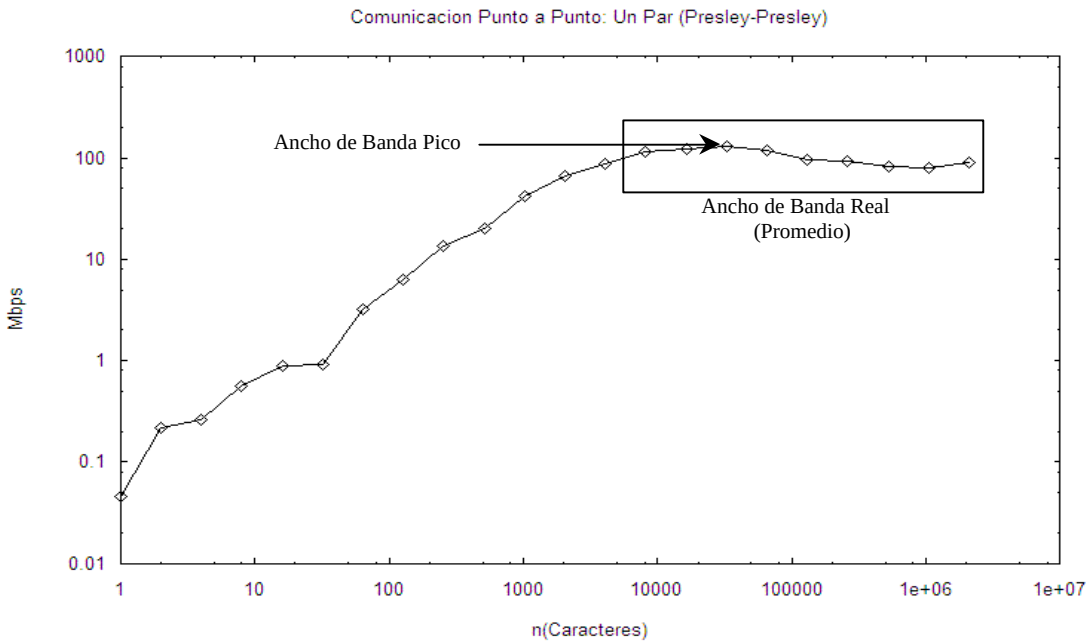


Figura 2.15 Prueba de Comunicación Punto a Punto: Un Par (Presley-Presley).

De las Figuras 2.13 y 2.15 obtenemos los parámetros *latencia* y *ancho de banda* respectivamente. Ajustando los datos de la Figura 2.13 a una recta por el método de mínimos cuadrados se obtiene la siguiente ecuación:

$$y(x) = 3.40044e\text{-}05 + (1.85402e\text{-}08x)$$

La *latencia* aproximada se obtiene del término constante de la ecuación de recta obtenida, por lo tanto:

Latencia = 34.0044µs

La Figura 2.15 muestra en ancho de banda real (en Mbps) obtenido como función del tamaño del mensaje. De aquí se obtiene el ancho de banda pico al comunicar dos procesadores que comparten memoria. El *ancho de banda pico* corresponde al máximo de la velocidad ideal de comunicación entre un par de procesos y el *ancho de banda real* corresponde a la velocidad real de comunicación entre un par de procesos considerando contención y congestión.

Estos valores se obtienen de los resultados de las Figura 2.15:

Ancho de Banda Pico = 16.1929235 MBps = 129.543388 Mbps
Ancho de Banda Real = 12.8578128 MBps = 102.862502 Mbps

En las Figuras 2.16 a 2.19 se muestran los resultados de la prueba de comunicación punto a punto entre un par de procesadores localizados en máquinas distintas (Elvis-Presley).

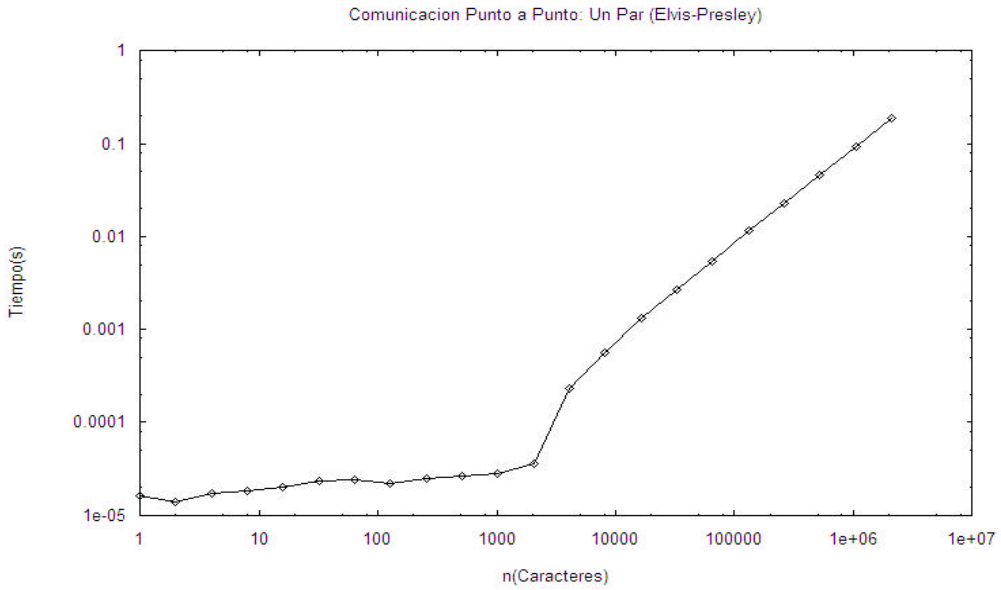


Figura 2.16 Prueba de Comunicación Punto a Punto: Un Par (Elvis-Presley).

Para poder apreciar de una mejor manera los resultados de la Figura 2.15, se dividió en dos partes como se muestra en las Figuras 2.17 y 2.18:

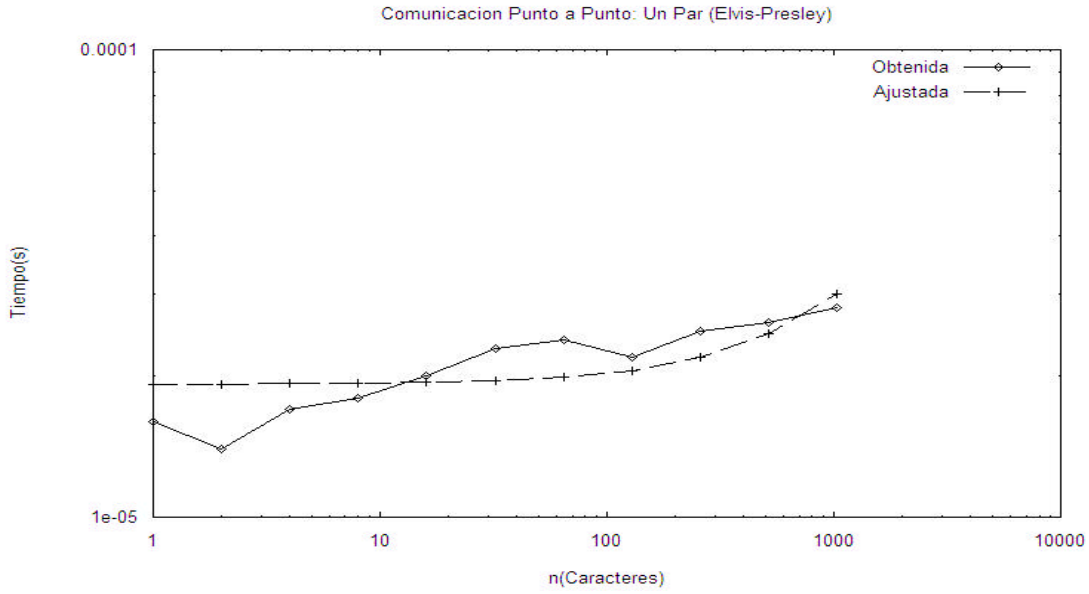


Figura 2.17 Prueba de Comunicación Punto a Punto: Un Par (Elvis-Presley).

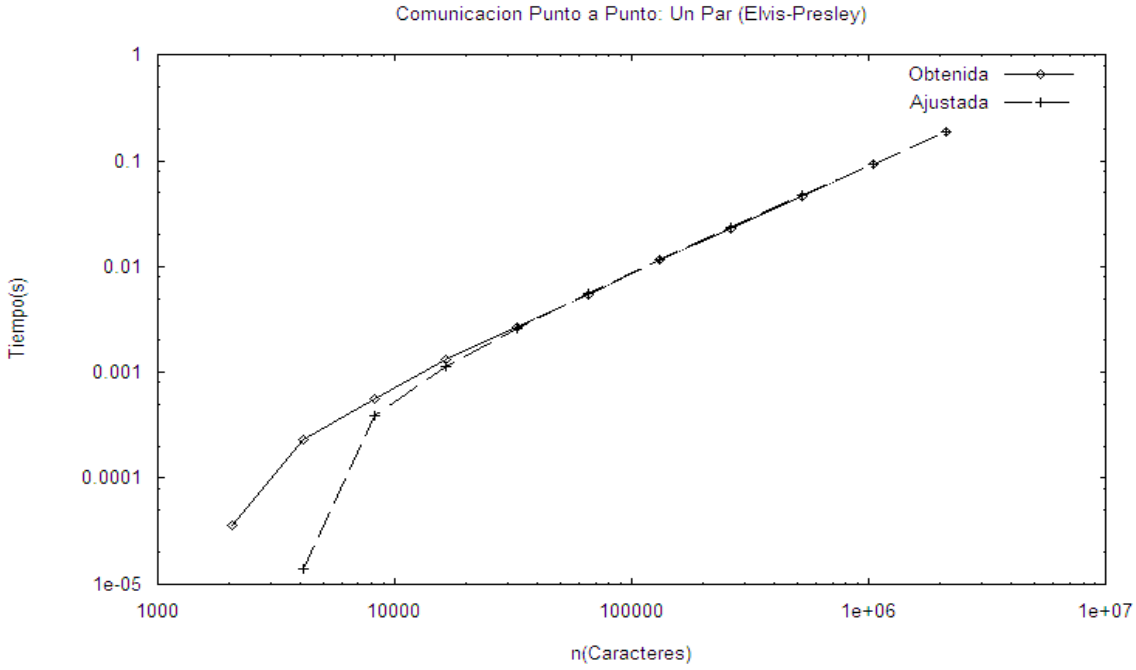


Figura 2.18 Prueba de Comunicación Punto a Punto: Un Par (Elvis-Presley).

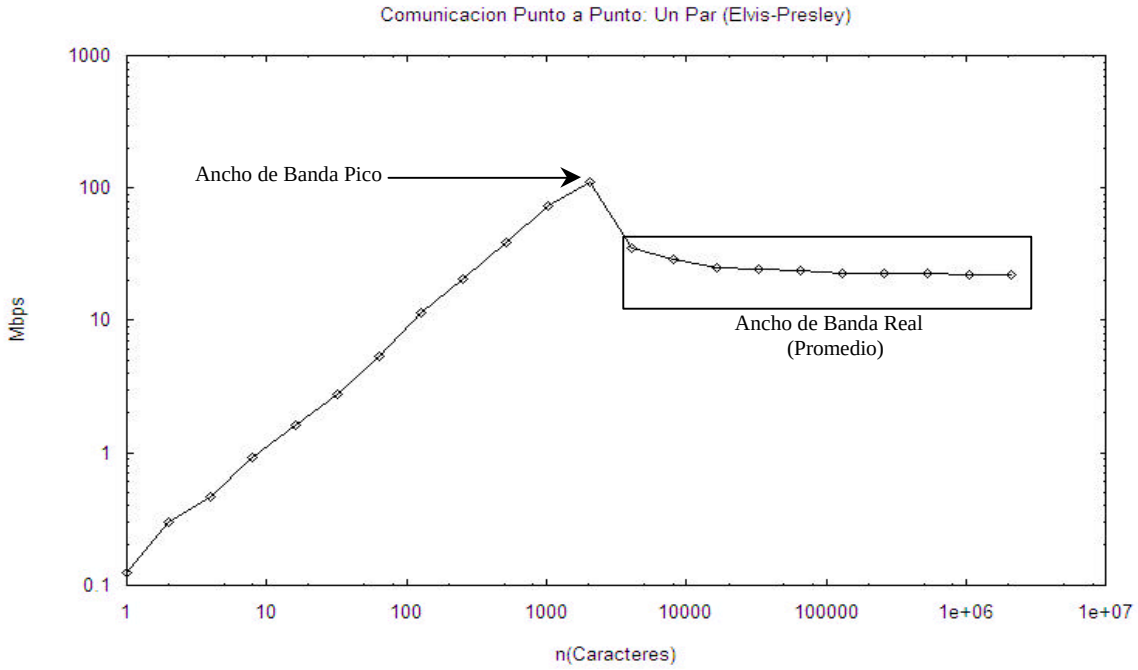


Figura 2.19 Prueba de Comunicación Punto a Punto: Un Par (Elvis-Presley).

De las Figuras 2.17 y 2.19 obtenemos los parámetros *latencia* y *ancho de banda* respectivamente. La ecuación de la recta obtenida por el método de mínimos cuadrados correspondiente a los datos de la Figura 2.17 es:

$$y(x) = 1.92263e\text{-}05 + (1.05084e\text{-}08)$$

La *latencia* aproximada se obtiene del término constante de la ecuación de recta obtenida, por lo tanto:

Latencia = 19.2263µs

El *ancho de banda pico* y el *ancho de banda real* se obtienen de los resultados de la Figura 2.19:

Ancho de Banda Pico = 14.104683 MBps = 112.837466 Mbps

Ancho de Banda Real = 3.131791 MBps = 25.0543292 Mbps

En las Figuras 2.20 a 2.23 se muestran los resultados de la prueba de comunicación punto a punto entre pares simultáneos de procesadores localizados en máquinas distintas (Elvis-Presley).

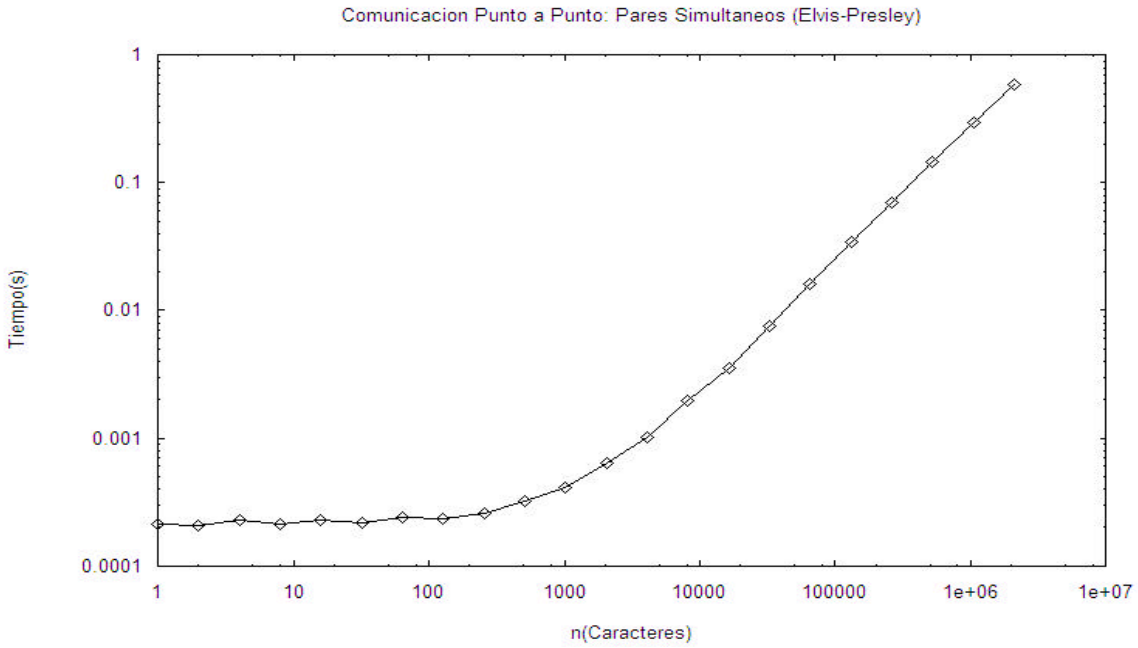


Figura 2.20 Prueba de Comunicación Punto a Punto: Varios Pares (Elvis-Presley).

Para poder apreciar de una mejor manera los resultados de la Figura 2.20, se dividió en dos partes como se muestra en las Figuras 2.21 y 2.22:

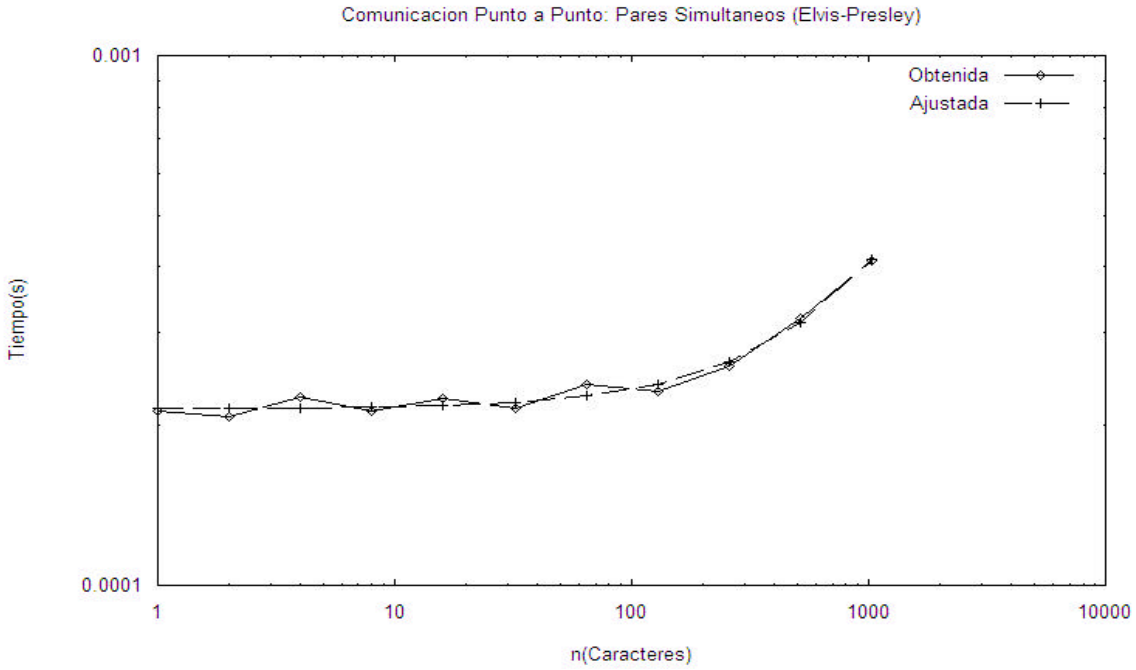


Figura 2.21 Prueba de Comunicación Punto a Punto: Varios Pares (Elvis-Presley).

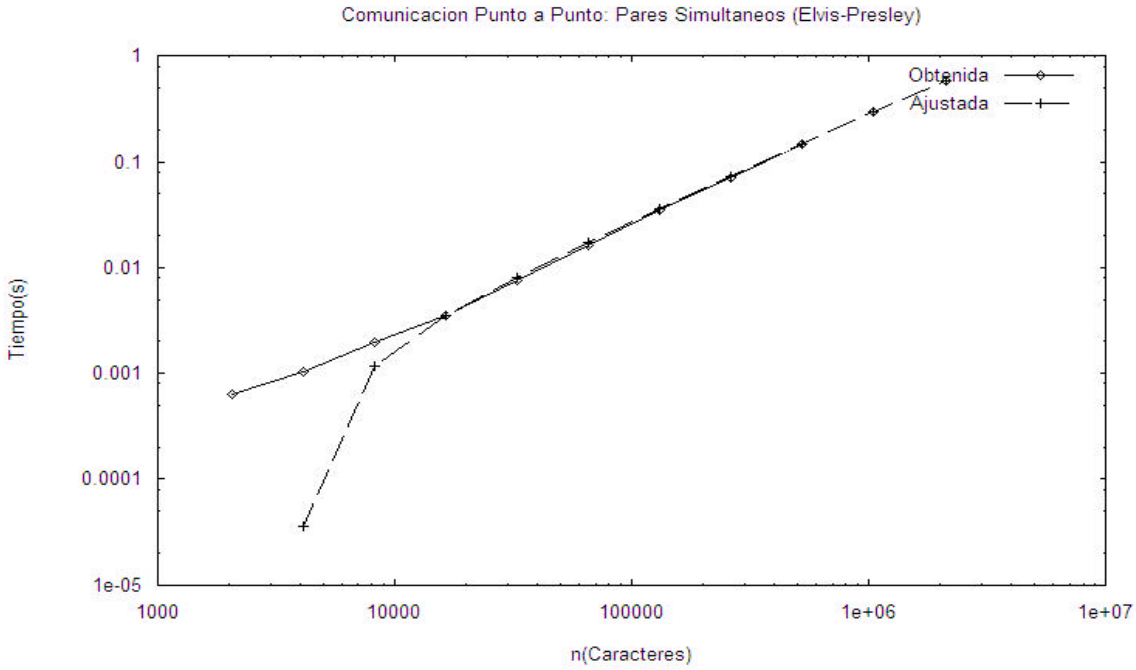


Figura 2.22 Prueba de Comunicación Punto a Punto: Varios Pares (Elvis-Presley).

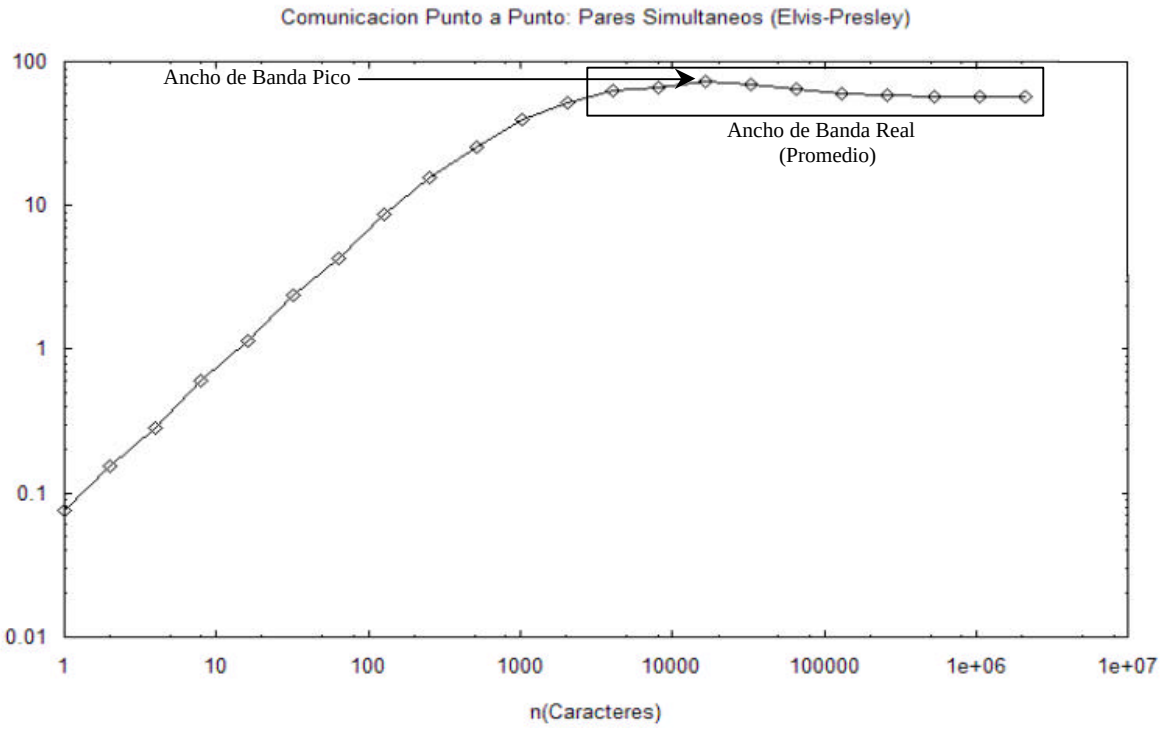


Figura 2.23 Prueba de Comunicación Punto a Punto: Varios Pares (Elvis-Presley).

De las Figuras 2.21 y 2.23 obtenemos los parámetros *latencia* y *ancho de banda* respectivamente. La ecuación de la recta correspondiente a los datos de la Figura 2.21 es:

$$y(x) = 0.0002154 + (1.92282e\text{-}07)$$

La *latencia* aproximada se obtiene del término constante de la ecuación de recta obtenida, por lo tanto:

Latencia = 215.4µs

El *ancho de banda pico* y el *ancho de banda real* se obtienen de los resultados de la Figura 2.23:

Ancho de Banda Pico = 9.168862 MBps = 73.3509 Mbps

Ancho de Banda Real = 7.5937535 MBps = 60.750028 Mbps

En la Tabla 2.5 se presenta una comparación de estos resultados con los resultados obtenidos al realizar las mismas pruebas en las siguientes arquitecturas. Todos los resultados presentados en la Tabla 2.5 corresponden a pruebas de comunicación entre procesadores localizados en máquinas diferentes.

1. Cluster de 8 Computadoras Pentium II 450 Mhz
   Switch FastEthernet 100 Mbps
   Hub FastEthernet 100 Mbps

2. SP2: 8 procesadores POWER2 Super Chip 120 Mhz
   Switch Lucent Technologies 122 Mbps
   Ethernet 10 Mbps

Descripción	Latencia	BW (un par)	BW (varios pares)
Switch 100 Mbps Beowulf	197µs	76.8 Mbps	62.4 Mbps
Hub 100 Mbps Beowulf	190µs	70.4 Mbps	16.9 Mbps
Switch SP2 976 Mbps SP2	321µs	78.4 Mbps	74.9 Mbps
Hub 10 Mbps SP2	455µs	7.6 Mbps	1.84 Mbps
Switch 100 Mbps Elvis-Presley	215.4µs	25.05 Mbps	60.75 Mbps

Tabla 2.5 Comparación de desempeño entre la máquina Elvis-Presley y otras arquitecturas.

El desempeño mostrado por las máquinas Elvis-Presley puede considerarse como bueno. A pesar de haber obtenido un tiempo de latencia relativamente alto y anchos de banda no muy satisfactorios, debe tomarse en cuenta que el switch utilizado para interconectar estas máquinas es compartido con otros servidores de la Sección de Computación, los cuales atienden a las diversas subredes de la misma.

Capítulo 3

Invocación Remota de Recursos Ejecutables del Laboratorio de Paralelismo

En este capítulo se aborda de manera muy resumida la forma de poder acceder de manera remota a las aplicaciones paralelas con las que cuenta el Laboratorio de Paralelismo de la Sección de Computación. Particularmente, esto involucra tareas como la administración de usuarios, vinculación de scripts CGI con las aplicaciones disponibles para su ejecución y la generación dinámica de los resultados obtenidos en forma de tablas de datos y gráficas.

3.1 Descripción General

El objetivo principal del sistema implementado es el de brindar al usuario un medio mediante el cual tenga acceso a un cierto número de aplicaciones paralelas residentes en el Laboratorio de Paralelismo las cuales pueda ejecutar de manera remota (ver Figura 3.1).

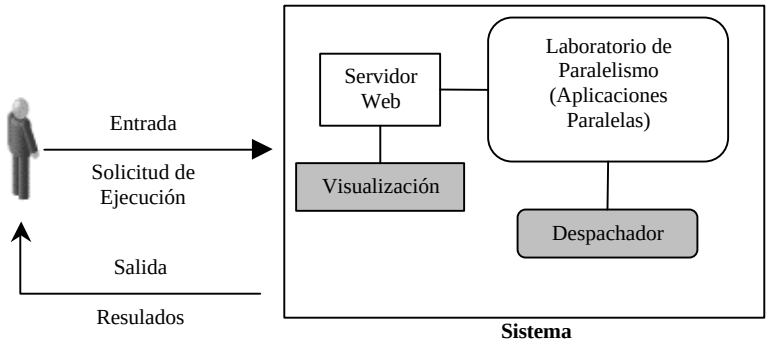


Figura 3.1 Visión general del sistema. El usuario solicita la ejecución de una de las aplicaciones del sistema obteniendo a la salida los resultados obtenidos.

El ciclo para la ejecución de manera remota de una de las aplicaciones con las que cuenta el sistema comprende cuatro pasos principales (ver Figura 3.2).

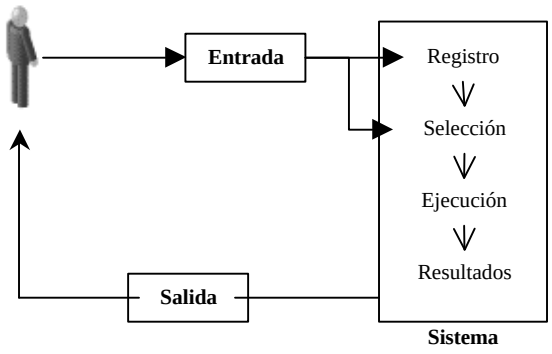


Figura 3.2 Ciclo de ejecución de una aplicación.

1. **Registro.** Proceso mediante el cual el usuario se identifica con el sistema para determinar si tiene o no acceso al mismo.
2. **Selección.** Una vez identificado satisfactoriamente, el usuario procede a elegir una de los recursos (aplicaciones paralelas) disponibles en el sistema.
3. **Ejecución.** El sistema ejecuta el recurso elegido por el usuario.
4. **Resultados.** El sistema muestra al usuario de forma gráfica (tablas y gráficas comparativas) los resultados obtenidos de la ejecución del recurso.

En el resto del capítulo se ha de suponer que todos los archivos que contienen los datos, formas y programas del sistema de acceso remoto al laboratorio de paralelismo se encuentran instalados a partir de la trayectoria base identificada como **$DIRECTORIO_RAIZ**.

3.2 Registro y Selección de Aplicaciones

Antes de que un usuario pueda seleccionar y ejecutar de manera remota algunos de los recursos del sistema debe de identificarse previamente utilizando un simple protocolo de login/password. En la Figura 3.3 se muestra el proceso de identificación que tiene que seguir el usuario para poder acceder a las aplicaciones del sistema.

Si el login y password introducidos por el usuarios son válidos, el sistema le brinda acceso al área de selección de aplicaciones (ver pasos 1, 2 y 3 de la Figura 3.3), en caso contrario le muestra una forma mediante la cual el usuario puede solicitar una cuenta para poder tener acceso al sistema (ver pasos 1 y 2’ de la Figura 3.3).

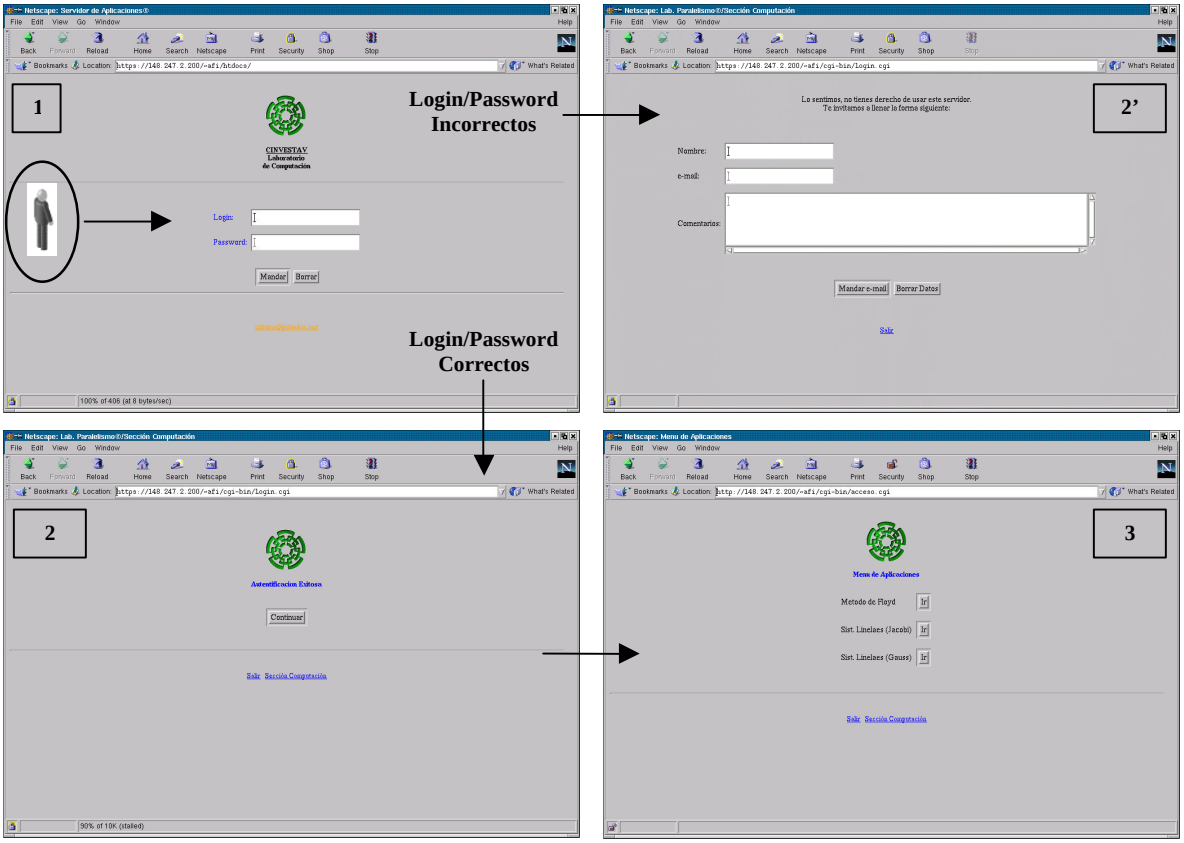


Figura 3.3 Proceso de registro y selección.

El protocolo para registro y selección de aplicación se establece utilizando Formas HTML y scripts CGI [14]. La siguiente sección explica de manera breve lo que son las Formas HTML.

3.2.1 Formas HTML

Una forma HTML es una sección de un documento que contiene elementos especiales llamados *controles* (checkboxes, botones radio (radio buttons), menús, etc.), y etiquetas para estos controles. El usuario generalmente "llena" el formulario modificando sus controles (insertando texto, seleccionando elementos de menú, etc.), antes de someter (submitting) la forma a un agente de procesamiento (e.g., a un servidor Web, a un servidor de correo, etc.)[7]. La Figura 3.4 muestra el código de una forma HTML sencilla, la cual contiene etiquetas, botones radio y botones push y la Figura 3.5 muestra la Forma HTML resultante.

forma.html

```
1: <FORM action="http://somesite.com/prog/adduser" method="post">
2:    <P>
3:    <LABEL for="firstname">First name: </LABEL>
4:          <INPUT type="text" id="firstname"><BR>
5:    <LABEL for="lastname">Last name: </LABEL>
6:          <INPUT type="text" id="lastname"><BR>
7:    <LABEL for="email">email: </LABEL>
8:          <INPUT type="text" id="email"><BR>
9:    <INPUT type="radio" name="sex" value="Male"> Male<BR>
10:    <INPUT type="radio" name="sex" value="Female"> Female<BR>
11:    <INPUT type="submit" value="Send"> <INPUT type="reset">
12:    </P>
13: </FORM>
```

Figura 3.4 Código HTML de una sencilla Forma HTML.

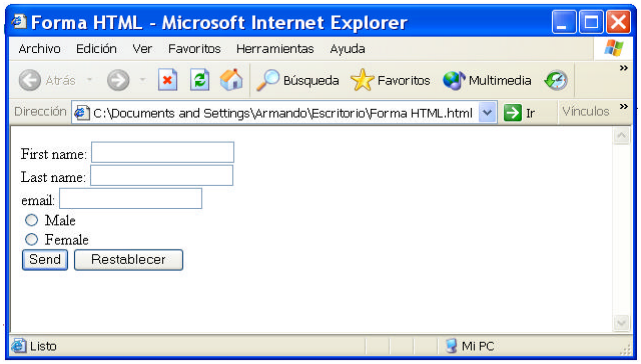


Figura 3.5 Forma HTML resultante del código de la Figura 3.4

Controles

El usuario interactúa con las formas a través de controles nombrados. El "nombre de control" de un control esta dado por su atributo *nombre* (por ejemplo, ver líneas 9 y 10 de la Figura 3.4). Cada control tiene un valor inicial y un valor actual, los cuales son cadenas de caracteres. En general, el "valor inicial" de un control puede especificarse con el atributo *valor* del elemento (por ejemplo, ver líneas 9 y 10 de la Figura 3.4).

Tipos de Controles

HTML define los siguientes tipos de control:

1. **Botones.** Existe tres tipos de botones:
    - botones submit: Cuando se activa, el botón submit somete (submit) una forma. Una forma puede contener más de uno de estos botones (ver Figura 3.5).
    - botones reset: Cuando se activa, un botón reset regresa a todos los controles a su valor inicial (ver Figura 3.5).
    - botones push: Estos botones no tienen un comportamiento establecido. Cada uno de estos botones puede estar asociados con un script el cual se ejecutará cada vez que el usuario lo presione.
2. **Checkboxes.** Los checkboxes (y los botones radio) son interruptores on/off los cuales el usuario puede cambiar su estado. Un interruptor está en "on" cuando el atributo *checked* del elemento de control esta puesto (set). Cuando una forma es sometida (submitted), solamente un control checkbox con su estado en "on" será tomado en cuenta.
3. **Botones radio.** Los botones radio son como los checkboxes excepto que pueden compartir un mismo nombre de control, el cual es mutuamente exclusivo, cuando uno de ellos se encuentra en "on" todos los demás serán puestos a "off" (ver Figura 3.5).

4. **Menús.** Los menús ofrecen al usuario una lista de opciones a elegir. El elemento *SELECT* crea un menú, en combinación con los elementos *OPTGROUP* y *OPTION*.
5. **Entrada de texto.** Existen dos tipos de controles que permitan al usuario introducir texto. El elemento *INPUT* crea una entrada de texto de una sola línea y un elemento *TEXTAREA* crea una entrada de texto de múltiples líneas (ver Figura 3.5).

3.2.2 Lista de Usuarios

Los usuarios que tienen acceso al sistema están representados mediante un archivo binario de texto con nombre **passwd**, el cual contiene el login y password de cada uno de ellos. Este archivo se encuentra localizado en la siguiente trayectoria de directorio:

$DIRECTORIO_RAIZ/data/*passwd*

Solamente el encargado del sistema tiene permisos de lectura/escritura sobre este archivo. Las operaciones que pueden ser realizadas por el encargado del sistema sobre este archivo son las siguientes:

1. Añadir usuarios
2. Buscar usuarios
3. Borrar usuarios
4. Cambiar password de usuarios
5. Ver los usuarios existentes

Estas operaciones las realiza utilizando el programa **adduser** que se encuentra localizado en la misma trayectoria de directorio que el archivo **passwd**. La utilización de este programa es realmente intuitiva como se puede apreciar en la Figura 3.6.

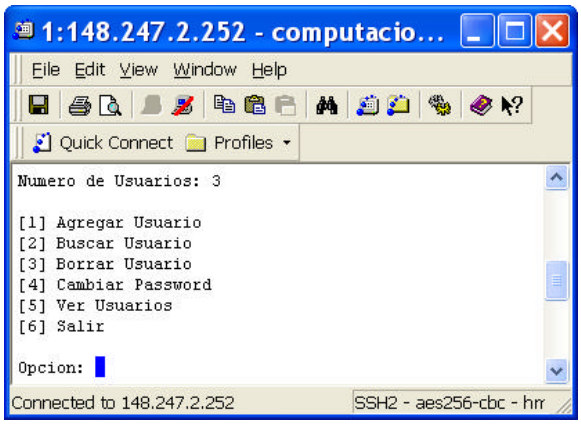


Figura 3.6 Programa de administración de la lista de usuarios del sistema.

3.2.3 Lista de Aplicaciones

Las aplicaciones paralelas con las que cuenta el sistema están representadas por una Forma HTML creada dinámicamente por un script CGI la cual las lista y muestra al usuario, como se puede apreciar en la Figura 3.7. Como se mencionó en la Sección 3.2 de este capítulo, solamente los usuarios que se hayan identificado satisfactoriamente en el sistema tienen acceso a esta forma.

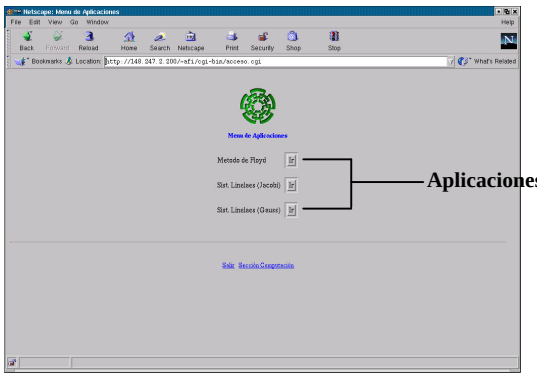


Figura 3.7 Forma HTML que muestra las aplicaciones paralelas con las que cuenta el sistema.

A su vez, cada una de las opciones mostradas en esta forma HTML está directamente relacionada con un script CGI el cual crea dinámicamente la correspondiente forma HTML encargada de recolectar los datos necesarios para la ejecución de la aplicación seleccionada, un ejemplo de esto se muestra en la Figura 3.8.

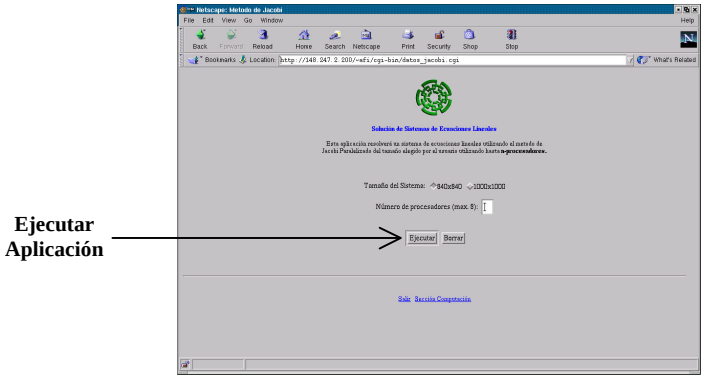


Figura 3.8 Forma HTML para la recolección de datos de aplicación.

Así mismo, cada una de las formas HTML que se utilizan para la recolección de datos de las aplicaciones, esta relacionada con un script CGI que es finalmente el encargado de mandar a invocar la ejecución de las mismas en el sistema. Este proceso se resume e ilustra en la Figura 3.9.

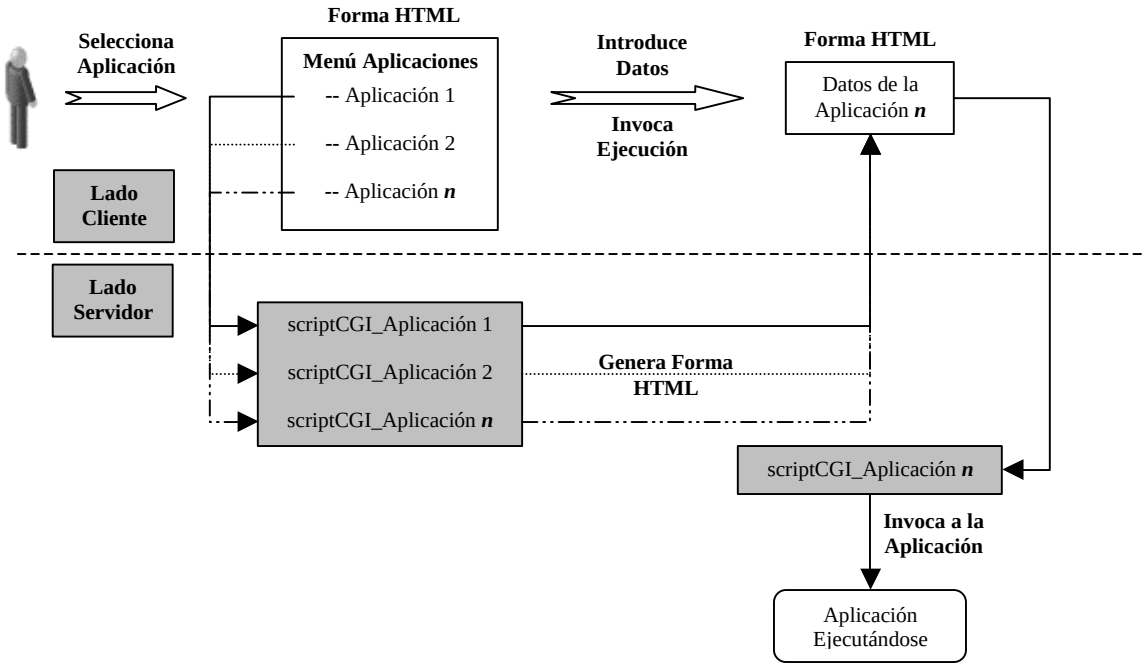


Figura 3.9 Proceso de selección y ejecución de una aplicación.

3.2.4 Agregando una Aplicación

Los archivos ejecutables de las aplicaciones del sistema, se encuentran localizados en la siguiente trayectoria de directorio:

$DIRECTORIO_RAIZ/cgi-bin/

Cuando se desea agregar una nueva aplicación y sus correspondientes scripts CGI (ver Figura 3.9) deben de añadirse dentro de esta trayectoria.

Ahora bien, una vez que se ha agregado la nueva aplicación dentro de la trayectoria especificada, se debe de modificar el código fuente del script CGI encargado de crear la página HTML que muestra al usuario el menú de aplicaciones del sistema (ver Figura 3.7), para que muestre la aplicación recién agregada. Este script CGI corresponde al archivo **acceso.cgi** localizado en la trayectoria de directorio ***$*DIRECTORIO_RAIZ***/cgi-bin/* y su código fuente **acceso.c** se encuentra localizado en ***$*DIRECTORIO_RAIZ** */codes/*. De esta manera se completa el proceso de agregar una nueva aplicación al sistema.En el Anexo A, se presenta el código de los archivos antes mencionados.

Las aplicaciones ya existentes en el sistema y las nuevas que se deseen agregar, deben cumplir con el requisito de haber sido programadas bajo el paradigma de *Descomposición de Dominio* de la programación paralela (ver Sección 2.2.1).

3.3 Ejecución de las Aplicaciones

Una vez que el usuario ha seleccionado una aplicación e introducido los datos necesarios para su ejecución, manda a invocar el script CGI correspondiente a través de la Forma HTML que sirve como interfaz con el sistema (ver Figura 3.8 y 3.9). Este script CGI se encarga de construir y ejecutar el comando necesario para que la aplicación seleccionada comience su ejecución.

El comando construido por el script, varía en relación a la aplicación seleccionada y al número de procesadores solicitados por el usuario, y tiene el siguiente formato:

mpirun –np *número de procesadores solicitados por el usuario* *aplicación seleccionada*

La Figura 3.10 ilustra de forma sencilla este proceso.

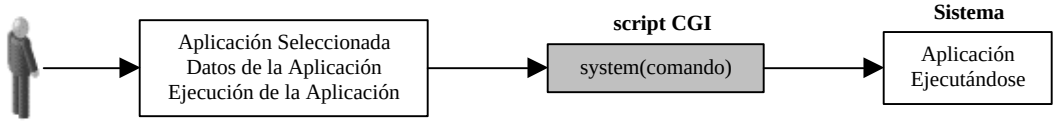


Figura 3.10 El script CGI se encarga de construir y ejecutar el comando que permite la ejecución de la aplicación en el sistema.

Este comando es ejecutado desde 1 hasta *n* veces dependiendo del número de procesadores total solicitados por el usuario. Por ejmplo, si el usuario solicitó ejecutar la aplicación con 3 procesadores, el script CGI construirá y ejecutará de forma sucesiva los siguientes comandos:

mpirun –np 1 *aplicación seleccionada*
mpirun –np 2 *aplicación seleccionada*
mpirun –np 3 *aplicación seleccionada*

Esto se hace con la finalidad de obtener los datos necesarios para mostrarlos posteriormente al usuario en forma de tablas y gráficas comparativas. Lo anterior se explica con detalle en la siguiente sección.

La descripción realizada aquí ha sido de manera muy resumida con el propósito de entender rápidamente la forma utilizada para ejecutar una aplicación paralela de manera remota. Esta descripción supone que un solo usuario está ejecutando una sola aplicación paralela variando secuencialmente el número de procesadores de 1 a *n*.

En realidad, el sistema esta diseñado para atender a varios usuarios a la vez con aplicaciones y parámetros diferentes. Por tanto, es necesario utilizar un despachador que organice las solicitudes de usuarios diferentes y las calendarice adecuadamente para su ejecución. Dado que este proceso es algo más complicado de describir, se ha trasladado su discusión para el Capítulo 4.

3.4 Resultados

Como se mencionó en la sección anterior, el script CGI encargado de ejecutar la aplicación seleccionada, ejecutará instancias de ésta utilizando desde 1 hasta *n*-procesadores solicitados por el usuario. Cada vez que una de estas instancias finaliza su ejecución escribe en un archivo de texto los datos siguientes:

- Número de procesadores utilizados para ejecutar la aplicación
- El tiempo que requirió para hacerlo

y se encuentra localizado en la siguiente trayectoria de directorio:

$DIRECTORIO_RAIZ/data/*archivo.dat*

Con la información contenida en este archivo, un nuevo script CGI se encarga de crear dinámicamente una página HTML la cual contiene tablas de datos y gráficas comparativas que representan dicha información. El proceso anterior se ilustra en la Figura 3.11.

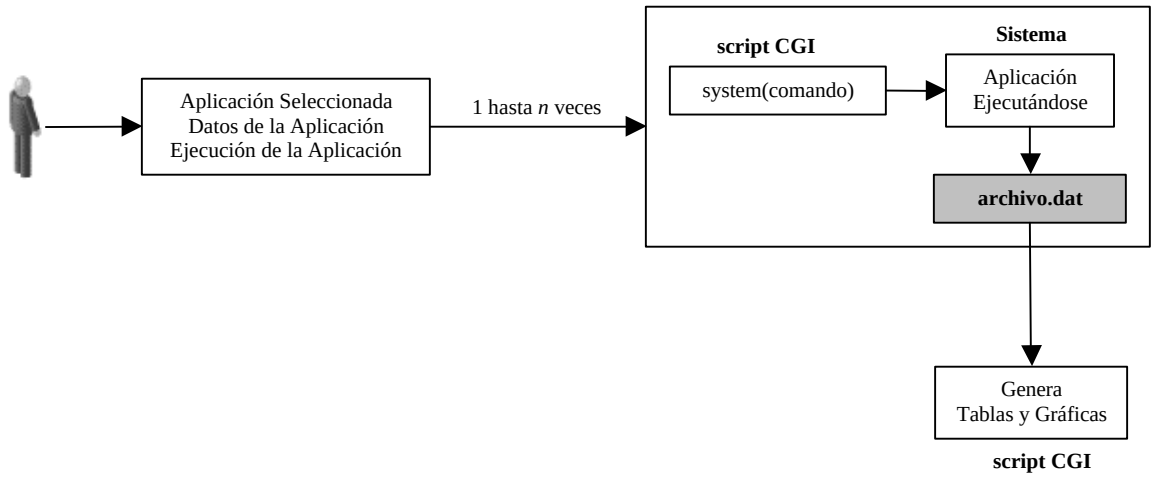


Figura 3.11 Generación de Tablas y Gráficas de Resultados.

El proceso de creación de resultados en realidad es más complejo. La descripción anterior es meramente ilustrativa con el fin de dar a entender de manera rápida la forma en la que el sistema genera los resultados y los muestra al usuario. Una descripción más completa y detallada de este proceso se presenta en el Capítulo 5.

3.5 Trayectorias importantes de la distribución

La Tabla 3.1 muestra un resumen de las diversas trayectorias y archivos mencionados en el capítulo.

Directorio	Descripción
DIRECTORIO_RAIZ/	Directorio a partir del cual las demás trayectorias están basadas. Es el directorio principal del sistema.
data/	En este directorio se encuentran archivos que contienen información útil para el sistema. Como lo son el archivo ***passwd***, los archivos ***usuario_nombre_de_la_aplicación.dat*** generados por las aplicaciones, y los scripts gnuplot para la creación de gráficas.
cgi-bin/	En este directorio se encuentran los diversos script CGI utilizados por el sistema como lo son ***acceso.cgi*** y ***graphics.cgi***, así como también los archivos ejecutables correspondientes a las aplicaciones paralelas.
codes/	En este directorio se encuentran localizados los diversos códigos fuentes correspondientes tanto a los scripts CGI como también a las aplicaciones paralelas.
pics/	En este directorio se encuentran localizadas las gráficas generadas por el script CGI de graficación.

Tabla 3.1 Resumen de trayectorias y archivos.

Capítulo 4

Calendarización de Programas Paralelos

En este capítulo se aborda el tema de la calendarización de procesos. Se describe cómo se puede utilizar eficientemente los recursos con los que cuenta el laboratorio de paralelismo al calendarizar de manera adecuada la ejecución de las diversas tareas que pueden ser solicitadas por uno o varios usuarios a la vez.

4.1 Introducción al Problema de Calendarización

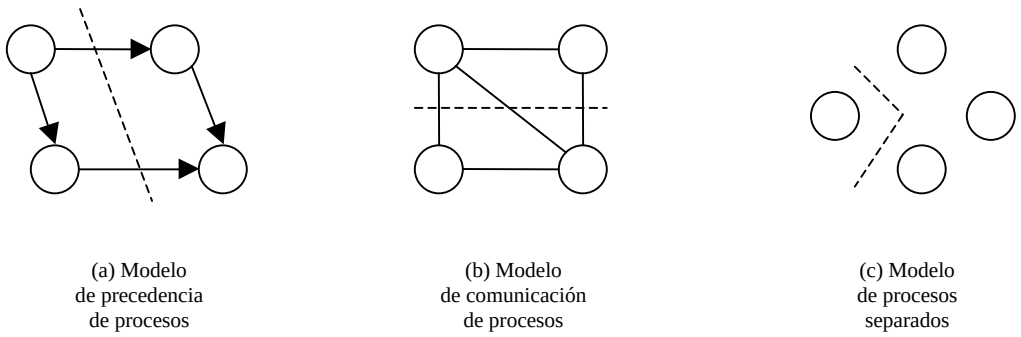
En una computadora secuencial, solamente existe en un momento dado, un solo programa ejecutándose. La computadora tiene solo un procesador el cual sirve un proceso a la vez. Un programa en ejecución es llamado *proceso*. Un proceso consiste de un programa, una pila y un conjunto de estructuras administrativas del sistema operativo. Cada proceso tiene un único identificador (*id*) el cual es usualmente un entero. Una computadora secuencial puede desempeñar *multiprogramación* mediante un rápido reasignamiento del CPU entre múltiples procesos, dando una apariencia ejecución simultánea. Este es el caso de los sistemas multi-usuarios, como por ejemplo Unix. Sin embargo, un verdadero paralelismo es provisto solamente por una computadora con múltiples CPU's.

Las facilidades de sincronización y comunicación son componentes esenciales para el soporte de la ejecución concurrente de procesos interactivos. Antes de su ejecución, los procesos necesitan ser calendarizados y designados con recursos. El objetivo primario de la calendarización es el de mejorar el desempeño de las métricas globales del sistema como lo son el tiempo de terminación de un proceso y la utilización del procesador.

La división de labores entre procesos paralelos para resolver un problema dado es llamada *calendarización*. La calendarización óptima se refiere a establecer una asignación de procesos a procesadores para realizar la ejecución general tan rápido como sea posible. Para mejorar la calendarización, usualmente se trata de *balancear* la cantidad de trabajo que cada proceso desempeña para evitar retardos causados por la dependencia entre procesos y minimizar la cantidad de actividades relacionadas con la calendarización en tiempo de ejecución.

4.1.1 Modelos de Aplicaciones

Los algoritmos paralelos y distribuidos son representados por conjuntos de múltiples procesos gobernados por reglas que regulan las interacciones entre ellos. La Figura 4.1 muestra un ejemplo de cómputo de un programa consistente de cuatro procesos mapeados, la interacción de procesos es expresada de manera diferente en cada uno de los modelos.



(a) Modelo de precedencia de procesos

(b) Modelo de comunicación de procesos

(c) Modelo de procesos separados

Figura 4.1 Modelo de descripción de aplicaciones paralelas.

4.1.1.1 Modelo de Precedencia de Procesos

Este modelo se muestra en la Figura 4.1(a), los procesos son representados por una *Gráfica Dirigida Acíclica* (DAG por sus siglas en Inglés). Los arcos denotan las relaciones de precedencia entre procesos y pueden incurrir en una sobrecarga de comunicación si los procesos conectados por un arco son mapeados hacia 2 procesadores diferentes. Este modelo es mejor aplicado a procesos concurrentes generados por constructores de lenguaje concurrente como *cobegin/coend* o *fork/join*. La calendarización de procesos aquí debe respetar las relaciones de precedencia y ejecutar un proceso únicamente hasta que sus procesos precedentes han concluido.

4.1.1.2 Modelo de Comunicación de Procesos

Este modelo se muestra en la Figura 4.1(b) la cual muestra un escenario diferente, donde los procesos son creados para coexistir y comunicarse asincrónicamente. Los bordes no dirigidos solo representan la necesidad de comunicación entre procesos. A partir de que el tiempo de ejecución no es explícito en el modelo, la meta de calendarización puede ser la de optimizar el costo total de comunicación y cómputo. Así una estrategia de calendarización para este modelo es asignar procesos que se comunican al mismo procesador o a procesadores que comparten ligas de comunicación. Los modelos de precedencia y comunicación de procesos son *modelos de interacción de procesos***.**

4.1.1.3 Modelo de Procesos Independientes

Este modelo se muestra en la Figura 4.1(c), la interacción entre procesos es implícita, y debemos asumir que los procesos pueden correr independientemente y completarse en un tiempo finito. Si los procesos son disjuntos, ellos son libres de moverse entre los procesadores, esto es, los procesos pueden moverse de nodos fuertemente cargados a nodos ociosos (si es que existen).

Uno puede ir un paso más allá para distribuir la carga de trabajo entre todos los nodos tan equitativamente como sea posible, ya sea estática o dinámicamente. Así las estrategias típicas de calendarización para este modelo son: compartición de carga (el hacer una distribución de carga estática) y *balance de carga* (el hacer una distribución de carga dinámica).

4.1.2 Calendarización Estática

La calendarización estática de procesos (o teoría de calendarización determinista) ha sido estudiada extensivamente. El problema es definido como la calendarización de un conjunto parcialmente ordenado de tareas en un sistema multiprocesador no-prioritario de procesadores idénticos para minimizar el tiempo de terminación general, la asignación de procesos a procesadores se realiza antes de que se ejecute cualquier proceso[2]. Esto permite generalizar la asignación de un proceso y un procesador reduciendo los tiempos de creación, sincronización y terminación de procesos, este tipo de calendarización se aborda por medio de heurísticas que encuentren soluciones válidas pero no óptimas (aproximadas), normalmente la decisión es centralizada y no adaptiva. Suposiciones comunes:

- No existen retrasos debido a comunicación o contención de la memoria.
- El tiempo de ejecución de cada tarea y las relaciones de precedencia entre tareas son fijas y conocidas de antemano.

Se pueden mencionar tres razones por las cuales la calendarización estática es interesante. Primera, la calendarización estática puede resultar algunas veces en tiempos de ejecución menores que la calendarización dinámica. Segunda, la calendarización estática permite la generación de un solo proceso por procesador, reduciendo la creación de procesos, sincronización, y sobrecarga de terminación. Tercera, la calendarización estática puede ser usada para predecir el aumento de producción realizado por un algoritmo paralelo en particular en una máquina objetivo, asumiendo que no ocurren preferencias entre procesos.

Como se mencionó en la Sección 4.1.2, una manera de ver un algoritmo paralelo es como una colección de tareas, algunas de las cuales deben ser completadas antes de que otras comiencen. En un *modelo determinista*, el tiempo de ejecución que necesita cada tarea y las relaciones de precedencia entre ellas son fijos y conocidos antes del tiempo de ejecución. Esta información puede ser representada por un gráfica dirigida llamado *gráfica de tarea*. Por ejemplo, considere la gráfica de tarea ilustrada en la Figura 4.2. Se tiene un conjunto de siete tareas y sus relaciones de precedencia.

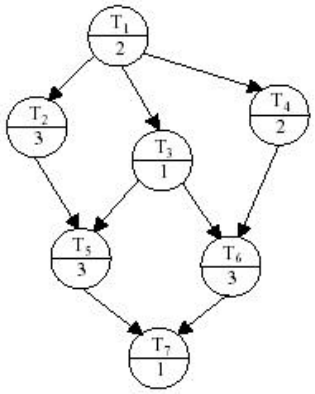


Figura 4.2. Ejemplo de una gráfica de tareas (DAG).

Una gráfica de tarea es una representación idealizada de la ejecución de un algoritmo paralelo, el cual ignora la varianza del tiempo de ejecución de las tareas debido a interrupciones, disputas por memoria compartida, etc. No obstante, los grafos de tareas proveen una base para la colocación estática de tareas en los procesadores.

Un *calendario* es una colocación de tareas en procesadores. Los calendarios son comúnmente ilustrados con cartas de Gantt. Una *carta de Gantt* indica el tiempo que cada tarea ocupa en su ejecución, así como también el procesador en el cual se ejecuta. Por ejemplo, la Figura 4.3 es una carta de Gantt de un calendario basado en una gráfica de tarea de la Figura 4.2. Una característica deseable de las cartas de Gantt es la de ilustrar gráficamente la *utilización* de cada procesador (el porcentaje de tiempo ocupado ejecutando tareas).

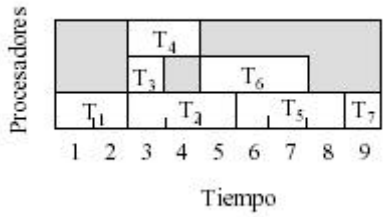


Figura 4.3 Carta de Gant

Dada una gráfica de tarea y un cierto número de procesadores, un **calendario óptimo** minimiza el tiempo total de ejecución. Algunos simples problemas de calendarización son solucionados en un tiempo polinomial, mientras que otros problemas, un poco más complejos, son intratables.

Por ejemplo, si todas las tareas toman una unidad de tiempo, y la gráfica es un bosque, esto es ninguna tarea tiene más de un predecesor, entonces existe un algoritmo de tiempo polinomial para encontrar un calendario óptimo.

Si todas las tareas toman una unidad de tiempo, y el número de procesadores son 2, entonces existe un algoritmo de tiempo polinomial para encontrar un calendario óptimo[3].

Si la longitud de las tareas varía del todo, o sí hay más de 2 procesadores, entonces el problema de encontrar un calendario óptimo es *NP-hard*[13], significando que los únicos algoritmos conocidos que encuentran un calendario óptimo requieren un tiempo exponencial en el peor caso.

Dada una lista de tareas ordenadas por su prioridad relativa, es posible asignar tareas a procesadores siempre asignando cada posible procesador a la primera tarea no asignada en la lista cuyas tareas predecesoras hayan terminado su ejecución. Este algoritmo de calendarización por lista fue propuesto por Graham y se formaliza a continuación.

4.1.3 Algoritmo de Calendarización por Lista de Graham

Sea $T = \{T_1, T_2, \ldots, T_n\}$ un conjunto de tareas. Sea $\mu : T \rightarrow (0, 8)$ sea una función que asocia un tiempo de ejecución con cada tarea. También dado un orden parcial ? en *T*. Sea *L* una lista de tareas en *T*.

Siempre que un procesador no tenga trabajo que hacer, una de las tareas en *L* es instantáneamente removida; esto es, una tarea no calendarizada cuyo predecesor bajo ? haya completado su ejecución. Si uno o más procesadores intentan simultáneamente ejecutar las mismas tareas, la tarea será asignada al procesador con el índice más bajo, y el otro procesador busca por alguna otra tarea disponible. La lista de tareas generada es la parte más importante del algoritmo para la optimización del problema[17][16].

4.1.4 Calendarización Dinámica

En la mayoría de las aplicaciones paralelas y distribuidas, no es realista suponer que se conocen de antemano las características de los procesos que se van a ejecutar. Sin saber los requerimientos de comunicación y cómputo, lo único que se puede hacer es establecer una calendarización adaptiva (dinámica) que permita que sus decisiones de asignación se hagan de manera local (descentralizada). Las metas de desempeño más intuitivas para la calendarización dinámica son el porcentaje de utilización del sistema y la cantidad de trabajos procesados por unidad de tiempo (*throughput*). Una simple estrategia heurística para lograr una alta utilización de un sistema es la de evitar tener procesadores ociosos tanto como sea posible.

4.2 Descripción General del Calendarizador de Programas Paralelos

Dado que se puede acceder al sistema desarrollado en esta tesis a través de Internet, éste debe ser capaz de atender múltiples solicitudes provenientes de usuarios diferentes que deseen tener acceso a alguno de sus recursos (aplicaciones paralelas). El sistema cuenta con un mecanismo que le permite hacer un uso eficiente de los recursos que son utilizados, este mecanismo esta constituido por un *Despachador de Mediano Plazo* y un *Despachador de Corto Plazo*.

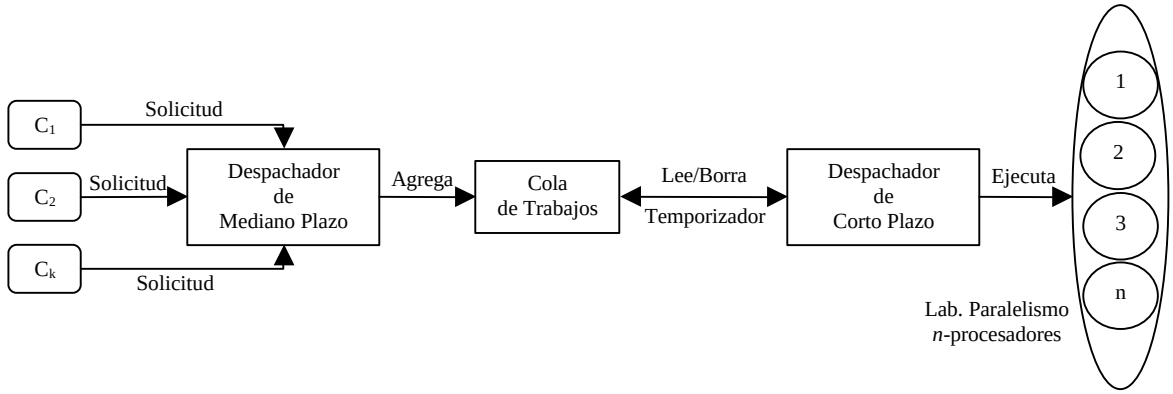


Figura 4.4 Despachador de Corto y Mediano Plazo

El propósito del Despachador de Mediano Plazo es el de estar continuamente "escuchando" solicitudes de cliente, y agregándolas a una Cola de Trabajos FIFO conforme vayan llegando, esperando ser atendidas por el Despachador de Corto Plazo (ver Figura 4.4). Cada vez que el Despachador de Mediano Plazo agrega una solicitud a la Cola de Trabajos contendrá estos 3 campos:

- *Usuario* – Login del usuario
- *Aplicación* – Nombre de la aplicación seleccionada
- *Número de Procesadores* – Que se desean utilizar

El Despachador de Corto Plazo cuenta con un temporizador, el cual periódicamente revisa la Cola de Trabajos en busca de solicitudes de cliente pendientes, toma la que se encuentre en el "fondo", la elimina de la cola, y la ejecuta, así sucesivamente hasta que ya no queden solicitudes de cliente pendientes.

En general se presentan 2 casos:

- Solicitud de n trabajos diferentes por el mismo usuario. Ejecución con 1, 2, 3,…, n procesadores. A este tipo de solicitudes se le denominará mediante el modelo *Un Cliente/Un Servidor*.
- k solicitudes de n_1, n_2,…, n_k trabajos diferentes. Ejecución con 1, 2, 3, …, n_j procesadores, $1 \leq j \leq k$. A este tipo de solicitudes se le denominará mediante el modelo *Varios Clientes/Un Servidor*

4.2.1 Modelo Un Cliente/Un Servidor

Por simplicidad se revisa primero cómo resolver el problema de atender la solicitud de un solo cliente (ver Figura 4.5). Aquí se supone que se trata de *n* trabajos a realizarse de manera independiente y no importa el orden en que se complete cada trabajo siempre y cuando los *n* sean realizados. En este caso, el sistema puede ejecutar de manera directa la solicitud del cliente; no es necesario utilizar una cola de trabajos ya que el cliente no puede solicitar una nueva ejecución hasta que la actual se haya completado.

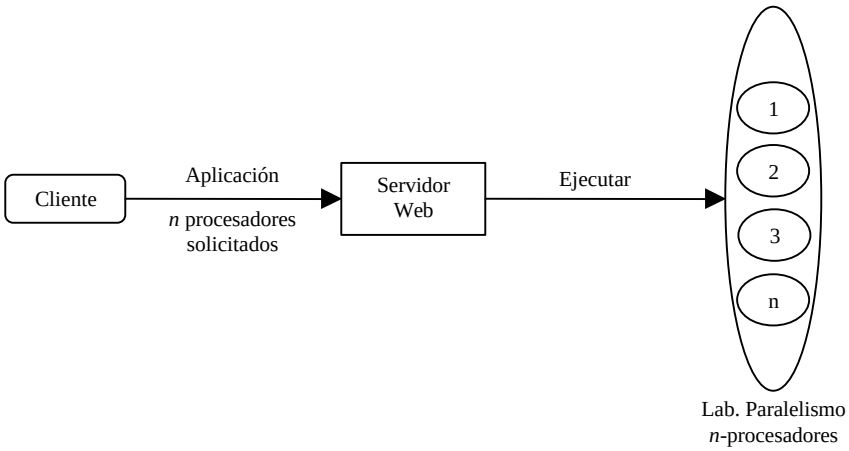


Figura 4.5 Modelo Un Cliente/Un Servidor

4.2.2 Modelo Varios Clientes/Un Servidor

En el caso en que varios clientes realizan solicitudes al sistema para la ejecución de recursos (aplicaciones paralelas), fue necesaria la implementación de un mecanismo (Cola de Trabajos, Despachador de Mediano y Corto Plazo) que coordinara la manera en la que las solicitudes fueran atendidas por el sistema (ver Figura 4.6).

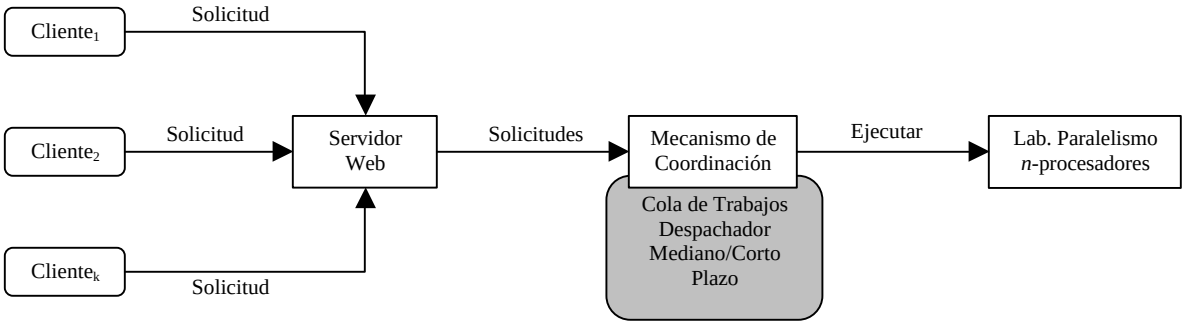


Figura 4.6 Modelo Varios Clientes/Un Servidor

En las secciones siguientes se explica con detalle los elementos que conforman este mecanismo de coordinación.

4.3 Algoritmo para el Despachador de Corto Plazo

El Despachador de Corto Plazo cumple con la misión de ejecutar las solicitudes de clientes pendientes en la Cola de Trabajos (ver Figura 4.7) haciendo un uso eficiente de los procesadores del sistema.

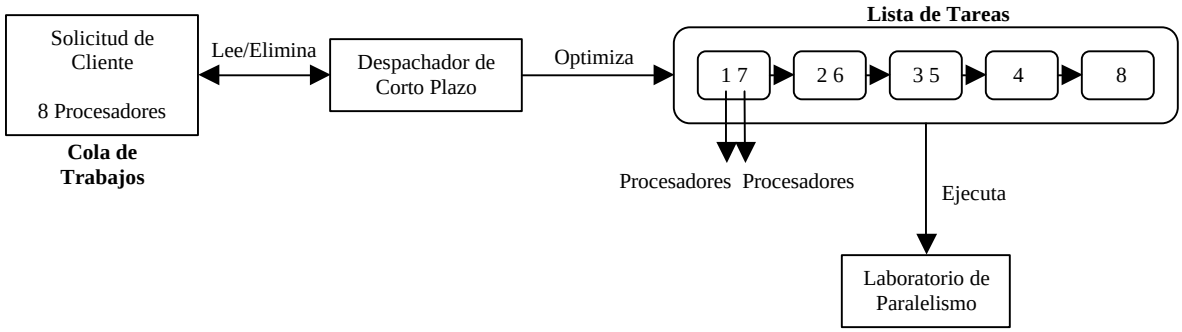


Figura 4.7 Despachador de Corto Plazo, lee y optimiza solicitudes de cliente para su ejecución.

```
1: N ← Número Total de Procesadores
2: int Lista[N] ← Lista de Tareas
3: int indice = 0

4: Inicializar Temporizador
5: Leer Cola de Trabajos

6: int PS ← Número de Procesadores Solicitados
7: int Mitad = PS/2
8: if PS = Número Par de Procesadores
9: {
10:       Desde i igual a uno hasta i menor o igual a Mitad

11:             Si i menor que Mitad
12:                   Agregar i a la Lista de Tareas
13:                   Aumentar i en uno
14:                   Agregar PS-1 a la Lista de Tareas;
15:                   Aumentar i en uno
                }

16:             Si i igual a Mitad
17:                   Agregar Mitad a la Lista de Tareas
18:                   Aumentar i en uno
19:             }
20:       }

21:       Agregar PS a la Lista de Tareas
22: }

23: else if PS = Número Impar de Procesadores
24:{
25:       Desde i igual a uno hasta i menor o igual a Mitad
26:             Agregar i a la Lista de Tareas
27:             Aumentar i en uno
28:             Agregar PS-1 a la Lista de Tareas
29:             Aumentar i en uno
30:       }

31:       Agregar PS a la Lista de Tareas
32: }

33: Leer Lista de Tareas
34: Ejecutar Instancias de la Lista de Tareas mientras no se exceda PS
```

Figura 4.8 Pseudocódigo del Despachador de Corto Plazo

Como se puede observar en la Figura 4.8, el Despachador de Corto Plazo con base en los datos obtenidos de la Cola de Trabajos crea una Lista de Tareas a ser ejecutadas, cada tarea cuenta con un número determinado de procesadores a utilizar. Al crear la lista, se toma en cuenta si el número de procesadores solicitados por el usuario es par o impar, siempre tratando de mantener ocupados la mayor cantidad de procesadores posible en cada ejecución de las tareas de la lista. La Figura 4.9 muestra un ejemplo de cómo el Despachador de Corto Plazo crea las correspondientes Listas de Tareas cuando el número de procesadores solicitados es par e impar respectivamente.

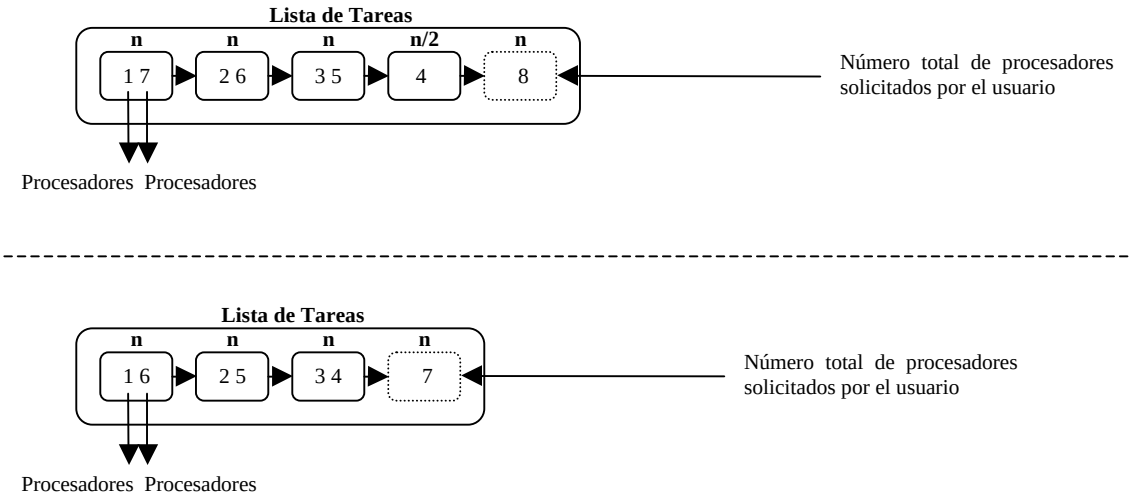


Figura 4.9 Creación de la Lista de Tareas por parte del Despachador de Corto Plazo. Observe como cada vez que se ejecuta una de las tareas de la lista se involucran *n-procesadores*, donde *n* es el número total de procesadores utilizados por el usuario. Esto cierto a excepción de cuando el número de procesadores solicitados es par, ya que la penúltima tarea de la lista ocupará *n/2* procesadores.

4.3.1 Medida de Rendimiento del Despachador de Corto Plazo

Como ya se mencionó, el propósito del despachador de Corto Plazo es el de mantener ocupados a la mayor cantidad posible de procesadores (máximo 8) en cada una de las ejecuciones de las tareas de la lista. Existen siete posibles listas que serán creadas por el despachador, dependiendo del número de procesadores solicitados por el usuario, las cuales se muestran a continuación en la Tabla 4.1 junto con el porcentaje de utilización de los procesadores brindado por cada una de ellas.

Procesadores solicitados	Lista de tareas	Porcentaje promedio de utilización
8	(1,7) (2,6) (3,5) (4) (8)	90%
7	(1,6) (2,5) (3,4) (7)	85.7%
6	(1,5) (4,3) (6)	87.5%
5	(1,4,2) (3,5)	93.75%
4	(1,3,2) (4)	62.5%
3	(1,2,3)	75%
2	(1,2)	37.5%
	Promedio total	76.25%

Tabla 4.1 Porcentaje de utilización brindado por las listas generadas por el despachador de corto plazo.

En cada una de las listas generadas por el despachador, el número de procesadores utilizados en cada instancia se encuentra encerrado entre paréntesis. Se utilizan la mayor cantidad posible de procesadores siempre y cuando la suma sea menor o igual a 8, que es el número total de procesadores con los que cuenta el sistema.

4.4 Descripción del Despachador de Mediano Plazo

Como se mencionó en la Sección 4.2 el propósito del Despachador de Mediano Plazo es el de estar continuamente “escuchando” solicitudes de cliente. Estas solicitudes las “escucha” el despachador utilizando un socket servidor que se encuentra continuamente atendiendo solicitudes de conexión para agregarlas a la Cola de Trabajos.

A diferencia de lo explicado en el Capítulo 3 acerca de la forma en que el sistema ejecuta las aplicaciones paralelas solicitadas por los clientes, con la implementación de los despachadores de corto y mediano plazo cada vez que un usuario solicita la ejecución de alguna aplicación su solicitud es agregada a la Cola de Trabajos para su futura ejecución, la Figura 4.10 ilustra de forma breve este proceso.

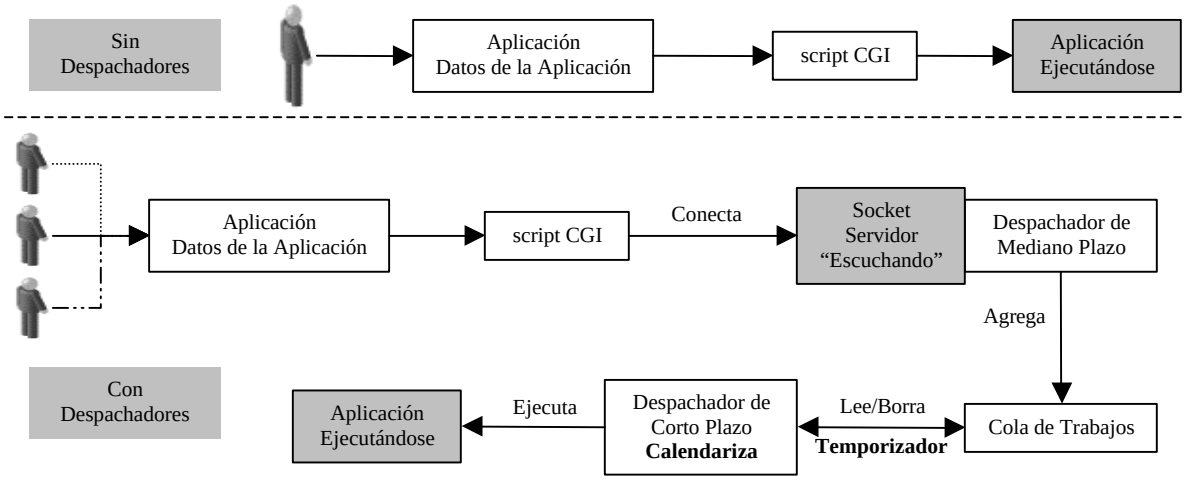


Figura 4.10 Proceso para la ejecución en el sistema de una aplicación paralela.

El pseudocódigo en la Figura 4.11 muestra el algoritmo utilizado en el Despachador de Mediano Plazo.

```
1: Archivo ← Cola de Trabajos

2: Inicializar void temporizador()

3: while(1) {
4:      Aceptar Conexiones → socket servidor

        /* Cuando se establece una conexión se atiende en un hilo de ejecución por separado */
5:      void agregar() ← Thread
6: }

7: void agregar()
8: {
9:      Bloquer Cola de Trabajos
10:     Cola de Trabajos ← Usuario
11:     Cola de Trabajos ← Aplicación
12:     Cola de Trabajos ← Número de Procesadores
13:     Desbloquear Cola de Trabajos
14: }

15: void temporizador()
16: {
17:     Bloquear Cola de Trabajos
18:     Leer Solicitud de Cliente
19:     Eliminar Solicitud de Cliente
20:     Desbloquear Cola de Trabajos
21:     Invocar Despachador de Corto Plazo
22: }
```

Figura 4.11 Pseudocódigo del Despachador de Mediano Plazo. El despachador crea un socket servidor para "escuchar" nuevas solicitudes de clientes remotos, cada nueva solicitud es atendida en un hilo de ejecución por separado (Thread) el cual agrega los datos de la nueva solicitud en la Cola de Trabajos. Por su parte el temporizador ejecutará cada una de las solicitudes en la Cola de Trabajos en intervalos de tiempo fijos.

4.5 Trabajo Relacionado

La calendarización y asignamiento es un tópico sumamente importante ya que una calendarización inapropiada de tareas puede fallar en la explotación del verdadero potencial del sistema y puede echar a perder la ganancia de la paralelización. En las siguientes secciones se hace referencia a diversos trabajos de investigación relacionados con este tópico.

4.5.1 Algoritmos de Calendarización Estática

La calendarización estática utiliza el conocimiento de las características del problema a resolver para alcanzar una solución global ó aproximada óptima. Aunque muchas personas han orientado su investigación de varias formas, comparten una idea en común: tomar un gráfica acíclica dirigida representando el programa paralelo como una entrada y calendarizandolo en procesadores de una máquina objetivo para minimizar el tiempo de completación. Este es un problema NP-completo es su forma general[18]. Sin embargo, varios algoritmos heurísticos que producen un desempeño satisfactorio han sido propuestos[31][19][15][29][13][32][1].

Un algoritmo secuencial es lento. La escalabilidad de la calendarización estática esta restringida debido a que se requiere una gran cantidad de espacio en memoria para almacenar la gráfica de tareas. Una solución natural a este problema es utilizar multiprocesadores para calendarizar tareas a multiprocesadores. De hecho, sin la paralelización del algoritmo de calendarización y ejecutándolo en una computadora paralela, un calendarizador escalable no es factible.

Un algoritmo de calendarización paralelo debe tener las siguientes características:

- **Alta calidad** – es capaz minimizar el tiempo de completación de un programa paralelo.
- **Baja complejidad** – es capaz de minimizar el tiempo para calendarizar un programa paralelo.

Estos dos requerimientos en general se contradicen entre si. Usualmente, un algoritmo de calendarización de alta calidad tiene una complejidad alta[20].

4.5.2 Herramientas para Calendarización de Procesos

Las herramientas de software poseen funcionalidades automatizadas para que la calendarización/mapeo pueda hacer la tarea de programación paralela más fácil. A pesar de que existe un amplio volumen de investigación sobre calendarización, la construcción de herramientas útiles de calendarización es algo que se ha abordado hasta últimas fechas. Una herramienta de calendarización debe de permitir al programador especificar el programa paralelo en una forma textual o gráfica, y después realizar un particionamiento automático y calendarización del programa. La herramienta debe de permitir también al usuario especificar la arquitectura objetivo. Una evaluación de desempeño y funciones de depuración son también altamente deseadas. Algunas herramientas proveen ambientes interactivos para la evaluación de desempeño de varias máquinas paralelas populares pero no generan ningún código ejecutable calendarizado.

Bajo la definición anterior, algunas herramientas proveen otras funcionalidades pero no pueden clasificarse como herramientas de calendarización. A continuación se mencionan algunas herramientas de calendarización.

4.5.2.1 Hypertool

Hypertool toma un programa secuencial particionado por el usuario como entrada y automáticamente asigna y calendariza las particiones a procesos. Las primitivas propias de sincronización son también automáticamente insertadas. Hypertool es una herramienta de generación de código a partir de que el programa del usuario es compilado en un programa paralelo para la computadora hipercubo iPSC/2 usando síntesis de código paralelo y técnicas de optimización. La herramienta también genera estimaciones de desempeño incluyendo tiempo de ejecución, comunicación y tiempos de suspensión para cada procesador así como también retardos de red para cada canal de comunicación.

4.5.2.2 PYRROS

PYRROS es un calendarizador de tiempo de ejecución y una herramienta de generación de código. Su entrada es una gráfica de tareas y el código secuencial en lenguaje C asociado. La salida es un calendario estático y un código paralelo en C para una arquitectura dada (la iPSC/2), PYRROS consiste de un lenguaje de gráfica de tarea con una interfaz para C, un sistema calendarizador el cual utiliza solamente el algoritmo DSC (Dominant Sequence Clustering), un visualizador gráfico basado en X-Window, y un generador de código. El lenguaje de gráfica de tarea permite al usuario definir programas particionados y datos. El sistema calendarizador es utilizado para agrupar la gráfica de tarea, realizar un mapeo balanceado de carga, y realizar ordenamiento de computación/comunicación. El visualizador gráfico es utilizado para desplegar gráficas de tareas y resultados de calendarización en forma de cartas de Gantt. El generador de código inserta primitivas de sincronización y realiza optimización de código paralelo para la máquina paralela objetivo.

4.5.2.3 Parallax

Parallax incorpora siete heurísticas de calendarización clásicas diseñadas en los años setentas, proveyendo un ambiente para que desarrolladores de programas paralelos puedan tener una noción de cómo los calendarios afectan el desempeño del programa en varias arquitecturas paralelas. Los usuarios deben de proveer el programa de entrada como una gráfica de tarea y estimar los tiempos de ejecución de las tareas. Los usuarios deben de expresar también la máquina objetivo como una gráfica de topología de interconexión. Entonces, Parallax genera calendarios en forma de cartas de Gantt, curvas de aceleración, y cartas de procesamiento y comunicación utilizando una interfaz X-Window. En adición, se muestra también una animación de la ejecución del programa simulado para ayudar a los desarrolladores a evaluar las diferencias entre las heurísticas de calendarización. Parallax, sin embargo, no genera ningún código ejecutable.

4.5.2.4 OREGAMI

OREGAMI esta diseñado para usarse conjuntamente con lenguajes de programación que soporten un modelo de comunicación, como los es OCCAM, C*, o C y FORTRAN con extensiones de comunicación. Como tal, es un conjunto de herramientas que incluyen un compilador LaRCS para compilar descripciones textuales de tareas de usuario en gráficas de tareas especializadas, las cuales son llamadas Gráficas Temporales de Comunicación (TCG por sus siglas en inglés). En adición, OREGAMI incluye una herramienta de mapeo para asignar tareas en una variedad de máquinas objetivo, y herramientas métricas para analizar y desplegar el desempeño. El conjunto de herramientas esta implementado en C para estaciones de trabajo SUN con una interfaz X-Window. Sin embargo, restricciones de precedencia entre tareas no son consideradas en OREGAMI. Más aún, no se genera ningún código objetivo. Así, como en el caso de Parallax, OREGAMI es más una herramienta de diseño para el desarrollo de programas paralelos.

4.5.2.5 PARSA

PARSA es una herramienta de software desarrollada para calendarización automática y particionamiento de programas secuenciales de usuario. PARSA consiste de cuatro componentes: una herramienta de especificación de aplicación, una herramienta de especificación de arquitectura, una herramienta de calendarización y particionamiento, y una herramienta de avaluación de desempeño. PARSA no genera ningún código objetivo. La herramienta de especificación de aplicación acepta un programa secuencial escrito en el lenguaje funcional SISAL y lo convierte en una DAG, la cual es representada en forma textual por lenguaje gráfica acíclico IF1. La herramienta de especificación de arquitectura permite al usuario especificar interactivamente el sistema objetivo en forma gráfica. El retraso de ejecución para cada tarea es también generado en base a la especificación de arquitectura

4.5.2.6 CASCH

CASCH aspira a ser un ambiente completo de programación paralela incluyendo paralelización, particionamiento, calendarización, mapeo, comunicación, sincronización, generación de código, y evaluación de desempeño. La paralelización es desempeñada por un compilador el cual automáticamente convierte aplicaciones secuenciales en códigos paralelos. El código paralelo es optimizado a través de una apropiada calendarización y mapeo, y es ejecutado en una máquina objetivo. CASCH provee una extensa librería de calendarización del estado del arte proveniente de literatura reciente. La librería de calendarización esta organizada en diferentes categorías las cuales son adecuadas para diferentes ambientes arquitectónicos.

Los algoritmos de calendarización y mapeo son utilizados para calendarizar la gráfica de tareas generada a partir del programa del usuario, el cual puede ser creado interactivamente o leído de un disco. Los pesos en los nodos y bordes de la gráfica de tareas son computados utilizando una base de datos que contiene los tiempos de varios cómputos, comunicación, y operaciones de E/S para diferentes máquinas. Estos tiempos son obtenidos a través de un benchmarking. Una característica atractiva de CASCH es su interfaz gráfica de usuario (GUI por sus siglas en Inglés) fácil de usar para el análisis de varios algoritmos de calendarización y mapeo utilizando gráficas de tareas generadas aleatoriamente, interactivamente, o directamente a partir de programas reales. El mejor calendario generado por un algoritmo puede ser utilizado por el generador de código para generar un programa paralelo para una máquina en particular, y el mismo proceso puede ser repetido para otra máquina.

4.5.2.7 Herramientas Comerciales

Existen solo unas pocas herramientas comerciales para la calendarización y paralelización de un programa. Algunos ejemplos incluyen a ATEXPERT por Cray Research; PARASPHERE por DEC; IPD por Intel; y PRISM por TMC. La mayoría de estas herramientas proveen ambientes desarrollo de software y depuración. Algunos de ellos proveen también herramientas de afinación de desempeño y otras facilidades de desarrollo de programas.

Capítulo 5

Visualización de la Ejecución Remota de Programas Paralelos

El propósito de este capítulo es hablar acerca de las facilidades visuales típicas que deben ofrecerse al usuario cuando requiere de la ejecución de uno de los recursos del sistema. Estas incluyen una herramienta que muestre el avance que la aplicación paralela vaya teniendo y una herramienta que muestre las actividades realizadas por el sistema durante la ejecución del recurso seleccionado por el usuario.

5.1 Facilidades de Visualización de Programas Paralelos

Cuando se ejecuta alguna aplicación es deseable que el usuario cuente con una retroalimentación la cual le muestre (de forma textual o gráfica) el avance que ésta vaya obteniendo con el fin de brindarle una noción acerca del desarrollo de la misma, por ejemplo, para indicarle si se esta ejecutando de manera correcta o no. En el caso particular de las aplicaciones paralelas, es deseable mostrar al usuario el avance que cada uno de los procesadores implicados en la ejecución del recurso vaya obteniendo, así como también las diversas actividades y procesos realizados por el sistema durante su ejecución (TOP – **T**able **O**f **P**rocess). La Figura 5.1 muestra un ejemplo de estas facilidades de visualización.

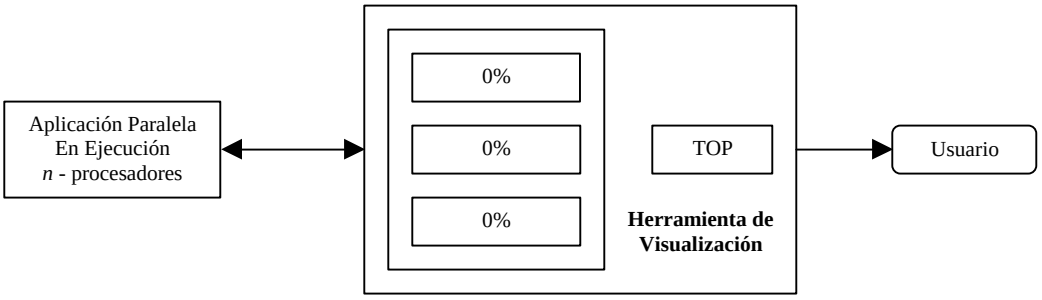


Figura 5.1 Facilidades de Visualización

En la actualidad existen este tipo de facilidades de visualización para 2 de las más importantes bibliotecas de paso de mensajes utilizadas para la programación de aplicaciones paralelas como lo son PVM y MPI: **XPVM** y **XMPI** respectivamente, las cuales se describen brevemente a continuación.

5.1.1 XPVM

XPVM[1] es un monitor y consola gráfica para PVM. Provee una interfaz gráfica para los comandos de consola PVM e información, junto con múltiples vistas animadas para el monitoreo de aplicaciones PVM. Estas vistas proveen información acerca de las interacciones entre tareas en un programa paralelo PVM, XPVM recolecta y despliega información en tiempo real.

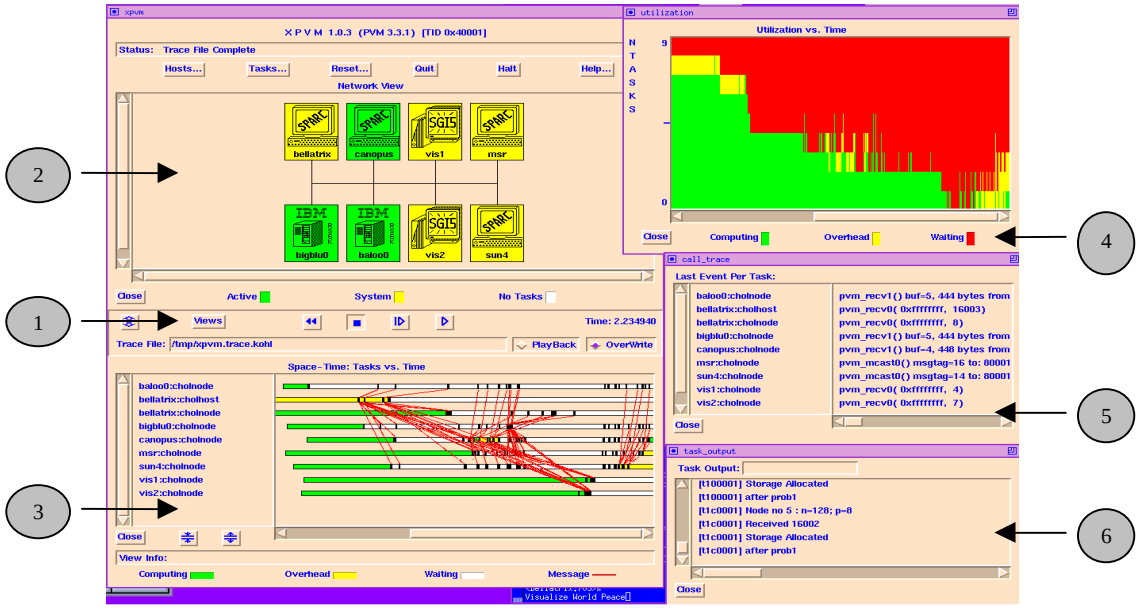


Figura 5.2 XPVM

En base a la figura 5.2, el botón *Views* (1) de la interfaz gráfica XPVM, permite al usuario abrir o cerrar cualquiera de las diferentes vistas que provee. Dos de las vistas más importantes son descritas a continuación:

- **Vista de Red (2)** - Despliega la configuración actual de la máquina virtual y la actividad de los hosts. Cada host es representado por un icono que incluye la arquitectura y nombre del host. Tales íconos toman distintos colores para indicar su estatus en la ejecución de PVM. Verde indica que al menos una tarea en ese host esta ejecutando trabajo útil. Amarillo indica que se están realizando llamadas a las rutinas del sistema de PVM. Cuando no hay tareas en un host, el icono se vuelve blanco.

[1] (http://www.netlib.org/utk/icl/xpvm/xpvm.html)

- **Vista Espacio-Tiempo (3) -** Permite ver la actividad que realizan las tareas PVM que están corriendo en la máquina virtual. Entre otras facilidades, esta vista nos permite ver el comportamiento de una tarea a lo largo del tiempo, así como la comunicación entre tareas.

5.1.2 XMPI

XMPI[2] provee una interfaz gráfica que despliega el estado de los procesos involucrados en una aplicación MPI. La información del estado de los procesos es obtenida a partir de una de dos fuentes, una aplicación en ejecución inicializada por XMPI o un archivo que contenga datos de traza. Cuando XMPI es inicializado, muestra una ventana en negro. Una vez que una aplicación es inicializada o un archivo de traza es cargado, la ventana muestra un grupo de hexágonos alineados, cada uno representando el estado de uno de los procesos MPI. Un símbolo de luz de semáforo indica si el proceso esta en ejecución o bloqueado (ver Figura 5.3).

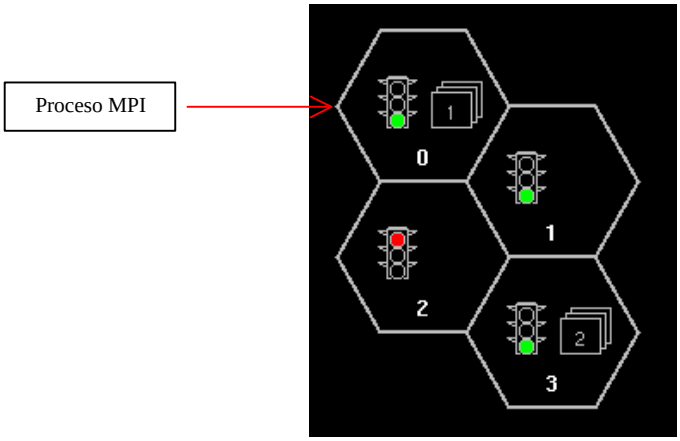


Figura 5.3 XMPI

Una ventana de proceso (como lo que se muestra en el Figura 5.4) con detalles MPI completos acerca de un proceso en particular y sus mensajes no recibidos, es desplegada mediante un clic del mouse sobre alguno de los hexágonos como los mostrados en la Figura 5.3.

[2] (http://www.lam-mpi.org/software/xmpi/)

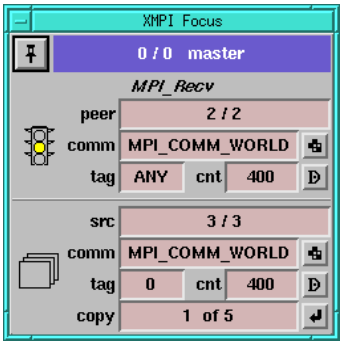


Figura 5.4 Ventana de detalles de proceso.

La característica fundamental de XPVM y XMPI es que están basadas en el modelo un cliente-un servidor para una solo aplicación. Así, en el caso de clientes múltiples, se despliega una consola para cada cliente con la información de su aplicación.

La información que presentan ambas herramientas proviene de archivos con la traza de la ejecución de la aplicación. Fundamentalmente, en la traza se captura toda la información referente a las interacciones de los procesos mediante el intercambio de mensajes.

XPVM y XMPI no proporcionan información acerca del desempeño del sistema al ejecutar una aplicación, información como la comparación entre el número de procesadores utilizados vs. tiempo ó curvas de aceleración, ya sea en forma de tablas comparativas, gráficas, ó una combinación de ambas. XPVM y XMPI no son adecuados para una interfaz entre un cliente y un servidor remotos por los requerimientos de comunicación que ambas herramientas exhiben.

5.2 Descripción General de la Herramienta de Visualización

El sistema cuenta con una herramienta de visualización la cual permite al usuario mostrarle de manera gráfica el avance en forma porcentual de cada uno de los procesadores involucrados en la ejecución de la aplicación seleccionada así como también una tabla de procesos (TOP) mostrándole de manera textual las diferentes acciones realizadas por el sistema durante la ejecución de la aplicación (ver Figura 5.5). Mediante lo anterior el usuario recibe información acerca del avance de su aplicación cuando ésta ya esta siendo ejecutada. Así también, conoce la lista de aplicaciones en espera de ser ejecutadas.

La información que la herramienta de visualización muestra al usuario esta basada en la lectura de archivos creados por la aplicación paralela y los despachadores de corto y mediano plazo. La generación y lectura de estos archivos se explica con más detalle en las secciones siguientes.

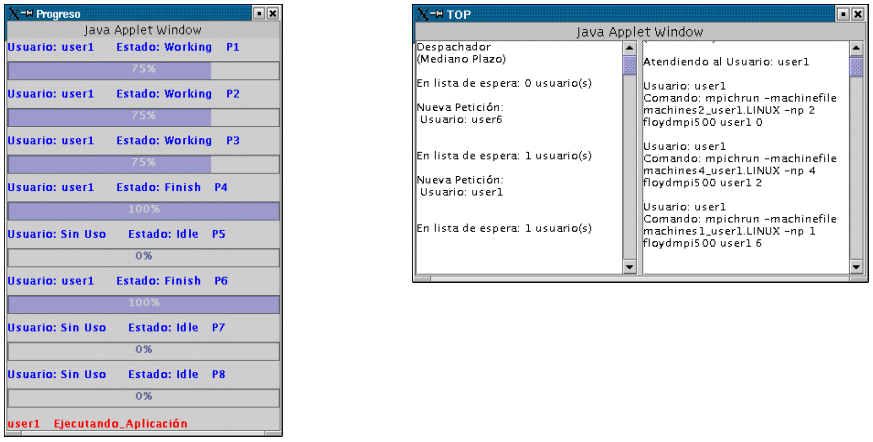


Figura 5.5 Herramienta de Visualización, conformada por barras de progreso y una tabla de procesos (TOP).

5.2.1 Archivo de Datos de las Aplicaciones Paralelas

De la misma manera que XPVM y XMPI guardan en archivos la traza de la ejecución de una aplicación paralela, la herramienta de visualización utiliza archivos en donde las aplicaciones almacenan información acerca del avance de una aplicación.

El archivo de datos generado por las aplicaciones paralelas con las que cuenta el sistema, contiene información acerca del progreso que cada uno de los procesadores involucrados en la ejecución de la aplicación va obteniendo. Estos archivos consisten de registros en donde se almacena la siguiente información:

- **usuario,** el nombre del usuario que solicito la aplicación
- **estado,** es el estado en que se encuentra el procesador: *comenzando*, *ejecutando*, *finalizado*
- **porcentaje,** El avance en forma porcentual que va obteniendo la aplicación (25, 50, 75 y 100%)

Los archivos rank*n*.dat se localizan en la siguiente trayectoria de directorio:

$ DIRECTORIO_RAIZ/htdocs/

Cuando se ejecuta una aplicación, cada uno de los procesadores generará uno de estos archivos rank**n**.dat (ver Figura 5.6).

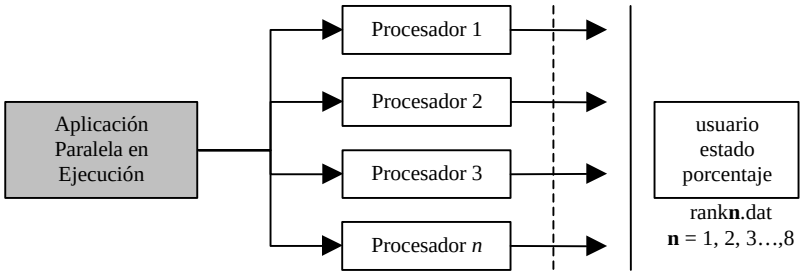


Figura 5.6 Cada uno de los procesadores involucrados escribe datos en el archivo rank*n*.dat que le corresponde.

Así, una aplicación para integrarse a este sistema debe estar diseñada para que en tales archivos vaya almacenando la información correspondiente. En otras palabras, los archivos deben de ser actualizados por la aplicación conforme esta vaya terminando de procesar datos, es decir, la porción de datos que le fue asignada a cada uno de los procesadores involucrados.

5.2.2 Archivo de Datos de los Despachadores de Corto y Mediano Plazo

De manera similar los despachadores de corto y mediano plazo escriben en un archivo los diferentes eventos y actividades que realizan. Estos archivos llamados **medianoPlazo.dat** y **cortoPlazo.dat**, constan de registros con los siguientes campos:

Despachador de Mediano Plazo (medianoPlazo.dat)

- **petición nueva** – Cada una de las nuevas solicitudes de cliente que le llegan al Despachador de Mediano Plazo
- **peticiones pendientes** – El número de peticiones de cliente pendientes para su ejecución

Despachador de Corto Plazo (cortoPlazo.dat)

- **Usuario** – Login del usuario del cual se esta atendiendo su aplicación solicitada
- **Comando** – Comando(s) que actualmente se esta(n) ejecutando para satisfacer la necesidad del cliente

Ambos archivos se encuentran localizados en la siguiente trayectoria de directorio:

$ DIRECTORIO_RAIZ/htdocs/

El proceso de actualización de tales archivos se ilustra en la Figura 5.7.

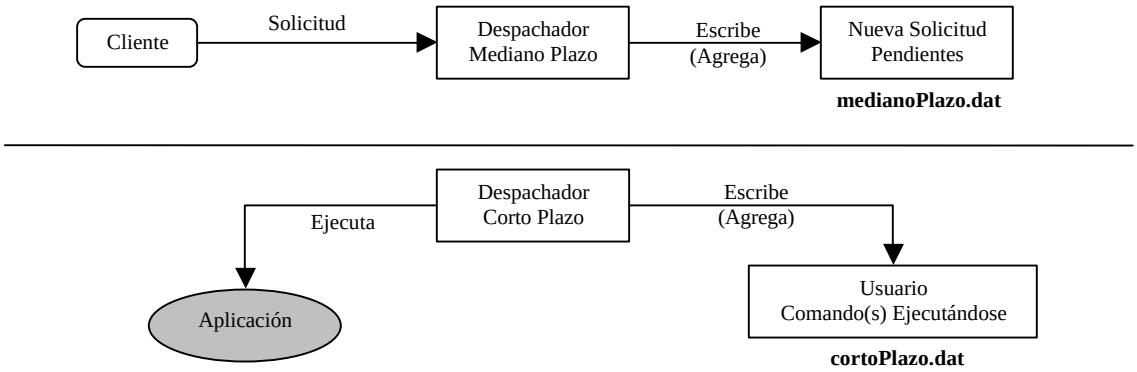


Figura 5.7 Los despachadores de corto y mediano plazo actualizan los datos de los archivos conforme se presentan nuevos eventos.

Los archivos de datos generados por las aplicaciones paralelas y los despachadores de corto y mediano plazo pueden ser leídos utilizando URLs (Universal Resource Location), Java provee las funciones de red necesarias para la lectura de archivos localizados en un servidor Web. Java también cuenta con las funciones necesarias para la creación de *temporizadores*, los cuales son comúnmente utilizados en aplicaciones para que periódicamente disparen acciones y monitoreen el progreso de operaciones[10]. Lo anterior se ilustra brevemente en la Figura 5.8.

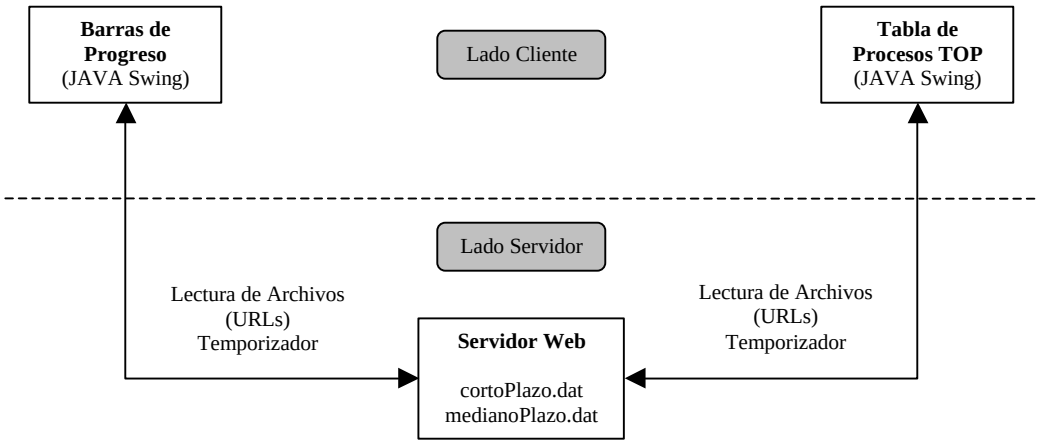


Figura 5.8 Herramienta de visualización del sistema.

De acuerdo a la Figura 5.8, la herramienta de visualización del sistema aprovecha las facilidades brindadas por el lenguaje Java (Swing, URLs, y Temporizadores) para mostrar al usuario de forma gráfica y textual (barras de progreso y TOP) la información contenida en los archivos generados por los despachadores de corto y mediano plazo.

5.3 Generación de Datos, Tablas y Gráficas

Cada vez que una de las aplicaciones del sistema finaliza su ejecución, escribe en un archivo de texto los datos siguientes: el *número de procesadores utilizado para ejecutar la aplicación* y *el tiempo que requirió para hacerlo*. El nombre de este archivo de texto, varía dependiendo del nombre de la aplicación seleccionada y del usuario que la seleccionó, y se encuentra localizado en la siguiente trayectoria de directorio:

$DIRECTORIO RAIZ/data/usuario_nombre_de_la_aplicación.dat

Con la información contenida en este archivo, un nuevo script CGI se encarga de crear dinámicamente una página HTML la cual contiene tablas de datos y gráficas comparativas que representan dicha información. Este script CGI es invocado por el usuario a través de una Forma HTML como la que se muestra en la Figura 5.9.

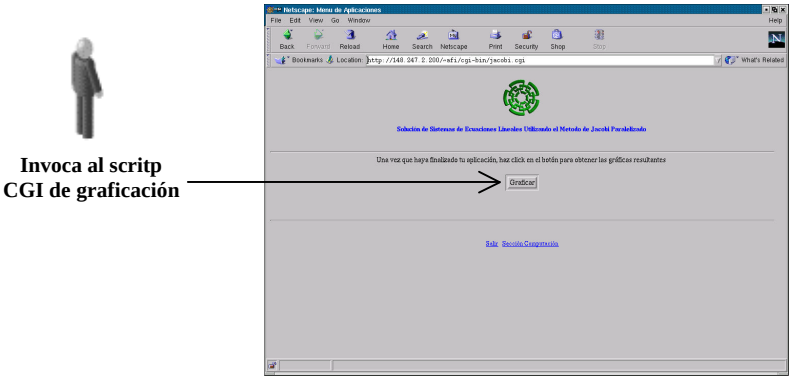


Figura 5.9 Una vez finalizada la ejecución de la aplicación seleccionada el usuario manda a invocar al script CGI de graficación.

Este script de graficación se encuentra representado en el sistema por el archivo **graphics.cgi** cuya trayectoria de directorio es:

$DIRECTORIO RAIZ/cgi-bin/*graphics.cgi*

La Figura 5.10 es un ejemplo de las tablas de datos y gráficas generados por el script de graficación.

Número de Procesadores	Tiempo
1	2.661
2	1.833
3	1.183
4	0.736

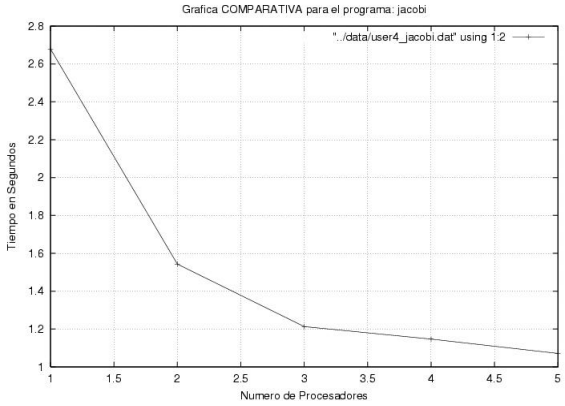


Figura 5.10 Ejemplo de tablas de datos y gráficas generados por el script de graficación.

La tabla de datos generada por el script es creada utilizando solamente código HTML, pero, su correspondiente gráfica es creada mediante la utilización de un programa de graficación y código HTML. El programa de graficación utilizado es **gnuplot**[3], es un software de libre distribución el cual es utilizado para graficar funciones y datos de puntos. En la siguiente sección se explica con más detalle como es que el script CGI hace uso de este programa de graficación, también se incluye una muy breve introducción acerca del mismo.

5.3.1 Gnuplot

Gnuplot es un programa manejado por líneas de comando para la graficación de funciones y ***datos de puntos***. Los archivos que contengan estos datos deben tenerlos ordenados en columnas de números. Las columnas deben estar separadas por espacios en blanco (tabulaciones o espacios), pueden contener comentarios (por ejemplo el nombre de las columnas) predecedidos por el símbolo #.

Un ejemplo de este tipo de archivo se muestra en la siguiente figura:

1	24.839
2	27.422
3	19.700
4	11.129
5	15.055
6	9.370
7	7.805
8	5.769

user2_gauss.dat

Figura 5.11 Ejemplo de un archivo de puntos de datos generados por el sistema para su graficación.

Gnuplot ofrece la ventaja de poder graficar un archivo de datos utilizando pequeños scripts, cada una de las líneas del script establece valores y parámetros que gnuplot tomará en cuenta al momento de hacer la graficación. Por ejemplo, la Figura 5.12 muestra un script gnuplot para la graficación del archivo *user2_gauss.dat* (ver Figura 5.11), y genera un archivo llamado *user2_gaussgraph.ps*.

[3] (http://www.gnuplot.info/)

```
#Script para la graficación de los resultados utilizando gnuplot
set title "Grafica COMPARATIVA para el programa: gauss"
set xlabel "Numero de Procesadores"
set ylabel "Tiempo en Segundos"
set grid
set data style linespoints
set autoscale
set output "../htdocs/pics/user2_gaussgraph.ps"
set terminal postscript
plot "../data/user2_gauss.dat" using 1:2
```

user2_gaussplot.p

Figura 5.12 Ejemplo de un script gnuplot generado por el sistema.

Una vez que se cuenta con estos dos archivos (el de datos y el script gnuplot) se manda a invocar al programa gnuplot para obtener la gráfica correspondiente:

```
$ gnuplot user2_gaussplot.p
```

Cuando el archivo **.ps** ha sido generado se procede a su conversión a formato **gif** utilizando el siguiente comando:

```
$ convert –rotate -90 user2_gaussgraph.ps user2_gaussgraph.gif
```

La siguiente figura muestra la gráfica generada por el programa gnuplot correspondientes a los archivos *user2_gauss.dat* y *user2_gaussplot.p*:

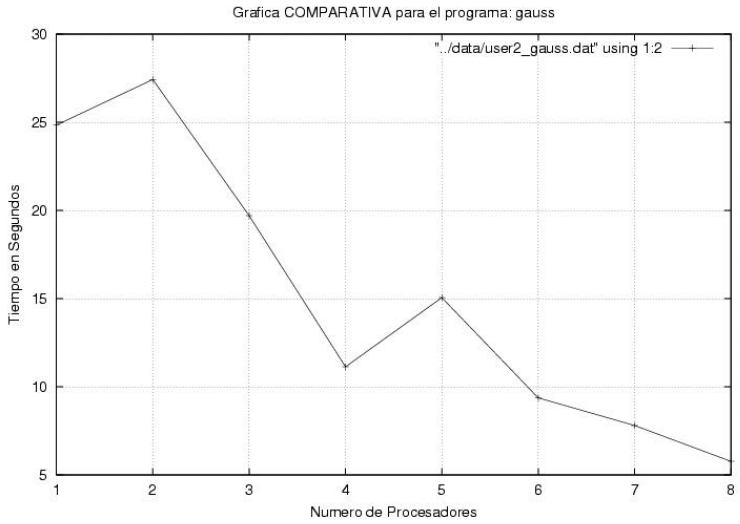


Figura 5.13 Gráfica generada por el programa gnuplot correspondientes a los archivos de las Figuras 5.11 y 5.12

Teniendo conocimiento de lo anterior, la generación de las gráficas mediante el script CGI correspondiente se realiza de manera muy similar. Como se mencionó anteriormente, el archivo en donde se escriben los resultados de los tiempos de ejecución de las aplicaciones sirve como base para que el script CGI construya las tablas de datos y las gráficas.

En base a este archivo el script CGI crea el correspondiente archivo ***usuario_nombre_de_la_aplicación.p*** que es el script gnuplot que le corresponde y lo guarda en la trayectoria de directorio *$DIRECTORIO RAIZ/data/*, una vez que el script CGI cuenta con estos dos archivos procede a ejecutar el comando:

```
$ gnuplot usuario_nombre_de_la_aplicacion.p
```

para finalmente obtener la gráfica. Las gráficas generadas por el script CGI se localizan en la trayectoria de directorio *$DIRECTORIO RAIZ/public_html/pics*. El proceso anterior se resume en la Figura 5.14.

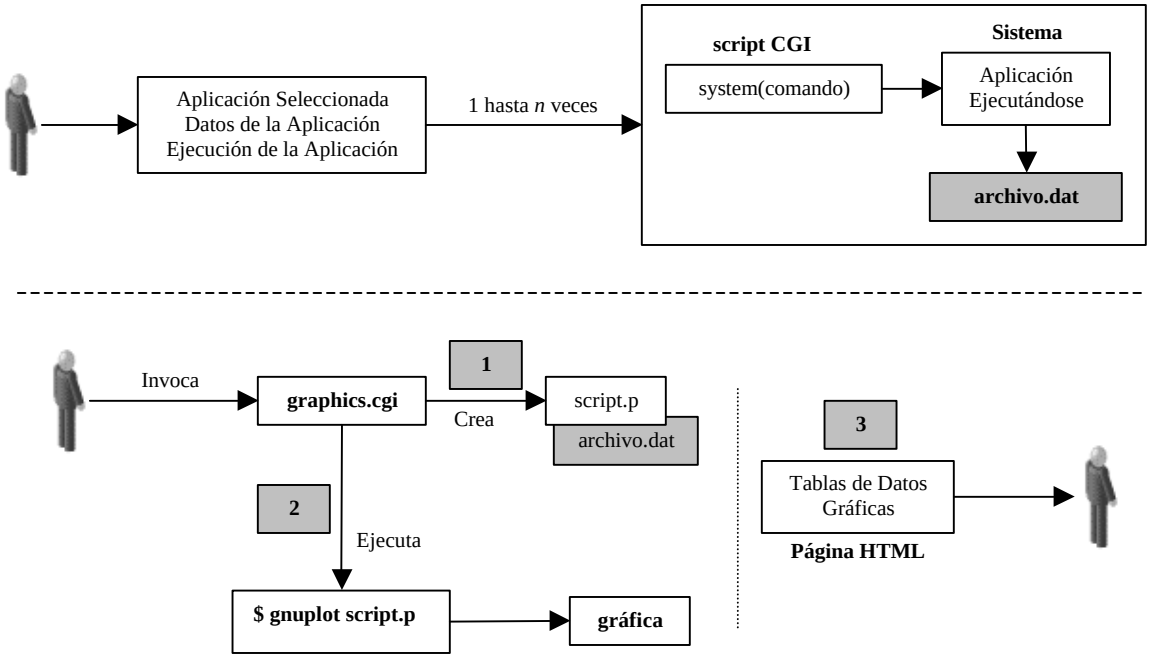


Figura 5.14 Proceso para la generación de tablas de datos y gráficas.

Capítulo 6

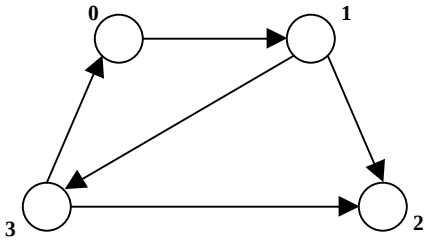
Resultados

En este capítulo se muestran los resultados obtenidos al ejecutar las aplicaciones paralelas con las que cuenta el sistema. Esto con el fin de comprobar que los diversos mecanismos que conforman el sistema (scripts CGI, los despachadores de corto y mediano plazo, la herramienta de visualización, y la tabla de procesos) funcionen de manera correcta y que se ha conseguido el objetivo planteado para esta tesis. Se encuentran disponibles las siguientes aplicaciones:

1. Algoritmo de Floyd.- Algoritmo para encontrar los caminos más cortos entre cualquier par de vértices en una gráfica.
2. Algoritmo de Jacobi.- Algoritmo para resolver sistemas de ecuaciones lineales.
3. Algoritmo de Gauss.- Algoritmo para resolver sistemas de ecuaciones lineales.

6.1 Algoritmos de Trayectorias más Cortas

El problema de la trayectoria más corta consiste en encontrar la trayectoria más corta entre todos los vértices de un grafo. Un grafo *G = (V, E)* abarca un conjunto de *V* vértices en *N*, **{v_i}**, y un conjunto $E \subseteq V \times v$ de bordes que los conectan. En una gráfica dirigida, cada borde tienen también una dirección, de tal forma los bordes **(v_{i1},v_j)** y (**v_{j1},v_i**), **i ? j**, son distintos. Un grafo puede representarse como una matriz de adyacencia *A* en la cual cada elemento *(i, j)* representa un borde entre el elemento *i* y *j*. $\mathbf{A_{ij}}$ = 1 si existe un borde **(v_{i1},v_j)**; de otra forma, $\mathbf{A_{ij}}$ = 0 (ver Figura 6.1).



	0	1	2	3
0	0	1	0	0
1	0	0	1	1
2	0	0	0	0
3	1	0	1	0

Figura 6.1 Un sencillo grafo dirigido (G), y su matriz adyacente (A).

Una *trayectoria* desde el vértice $\mathbf{v_i}$ al vértice $\mathbf{v_j}$ es una secuencia de nodos $\mathbf{(v_{i1},v_k)}$, $\mathbf{(v_{k1},v_i)}$, …, $\mathbf{(v_{m1},v_j)}$ en *E* en la cual ningún vértice aparece más de una vez. Por ejemplo, (1,3), (3,0) es una trayectoria del vértice 1 al vértice 0 en la Figura 6.1. La trayectoria mas corta entre dos vértices $\mathbf{v_i}$ y $\mathbf{v_j}$ en un grafo es la trayectoria que tiene menos bordes. El problema de la trayectoria más corta de *un-solo-origen* requiere que se encuentre la trayectoria más corta entre un vértice y todos los demás vértices de un grafo. El problema de la trayectoria más corta de *todos-los-pares* requiere que se encuentre la trayectoria más corta entre todos los pares de vértices de un grafo.

Uno de los algoritmos más utilizados para la solución de este problema es el algoritmo de Floyd-Warshall, en el cual, los bordes están representados como una matriz de adyacencia *D* donde el elemento d_{ij} es el peso del borde dirigido de *i* a *j*. (Para toda *i*, d_{ij}=0, y d_{ij}=8 sino existe un borde de *i* a *j*. El algoritmo itera una vez por cada vértice, actualizando continuamente la matriz de adyacencia *D*. En la k-ésima iteración, el algoritmo relaciona (relaxes) todos los vértices del grafo con respecto al vértice *k* de tal forma que la matriz de adyacencia *D* contiene todas las parejas de trayectorias más cortas que solamente utilizan los primeros *k*-vértices como intermediarios. El borde relacionado (relax) d_{ij} con respecto al vértice *k* se establece de la siguiente manera:

$$d_{ij} = \min\left(d_{ij}, d_{ik} + d_{kj}\right)$$

El algoritmo corre en un tiempo $O\left(|V|^3\right)$ ya que todos los pares *i*, *j* deben actualizarse en cada iteración.

La Figura 6.2 muestra el pseudocódigo del algoritmo de Floyd-Warshall.

```
Entrada: la matriz de adyacencia Dij
for k = 1 to V
  for i = 1 to V
    for j = 1 to V
      Dij = min(Dij, Dik+Dkj)
Salida: contiene la trayectoria más corta de i a j
```

Figura 6.2 Pseudocódigo del algoritmo de Floyd-Warshall

Una forma sencilla de paralelizar este algoritmo es dándole a cada procesador una igual porción contigua de filas de la matriz de adyacencia inicial (en la cual la entrada d_{ij} representa el costo de ir directamente del vértice *i* al vértice *j*). Cada una de las $k= 1…|V|$ iteraciones realiza lo siguiente:

- El procesador que posee la fila *k* realiza una operación de broadcast para comunicársela a todos los demás procesadores.
- Cada procesador recibe esta fila y relaciona cada una de sus entradas con respecto al vértice *k*. Con la fila *k* y la fila *i* disponibles, un procesador puede relacionar d_{ij} porque puede referir a d_{ik} (debido a que esta entrada es una fila *i*) y a d_{kj}.

La Figura 6.3 ilustra lo anterior de forma breve.

```
Procesador Pi posee sus propias filas |V|/p
for k = 1 to |V|
  El procesador Pk comunica a los demás procesadores
  la fila Dkj
  Todos los procesadores computan
    Dij=min(Dij, Dik+Dkj)
```

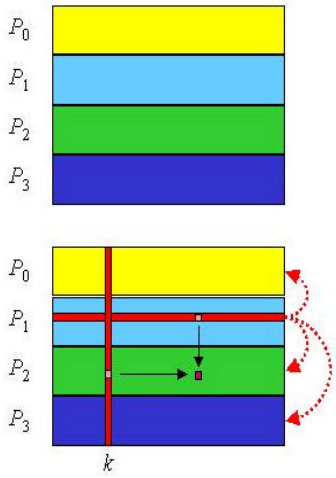


Figura 6.3 Pseudocódigo del algoritmo paralelo de Floyd-Warshall

6.1.1 Resultados Obtenidos

Los siguientes resultados fueron obtenidos al ejecutar la aplicación paralela que implementa el algoritmo de Floyd-Warshall la cual representa un grafo de 1000 vértices, utilizando 8 procesadores para su procesamiento.

Número de Procesadores	Tiempo (segundos)
1	119.815
2	100.299
3	80.903
4	41.862
5	52.338
6	26.701
7	22.605
8	16.005

Tabla 6.1 Resultados Obtenidos para el Algoritmo de Floyd-Warshall

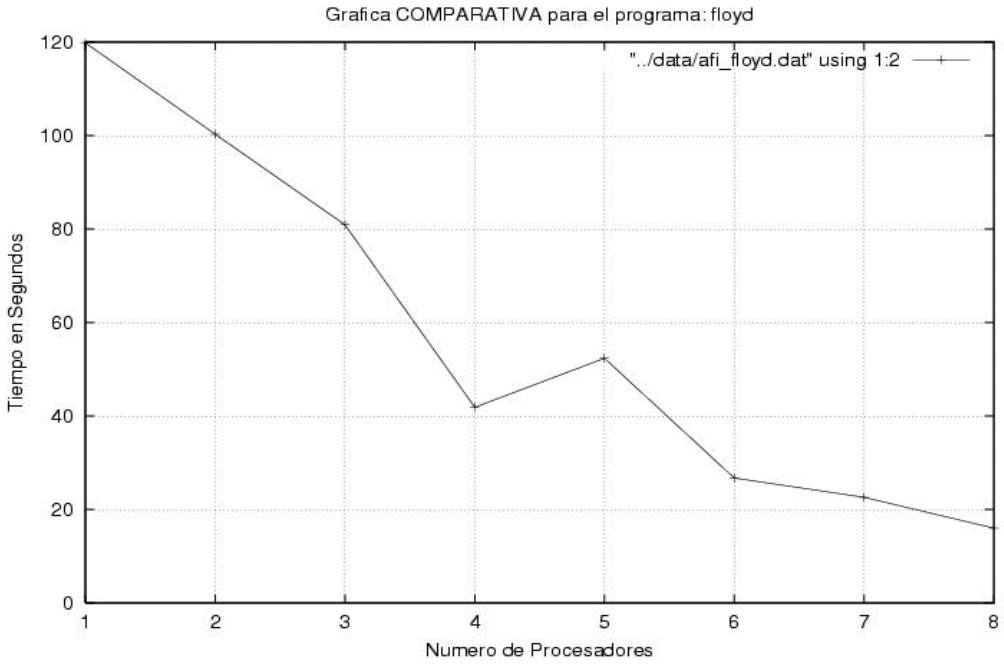


Figura 6.4 Gráfica de Resultados para el Algoritmo de Floyd-Warshall.

Número de Procesadores	Aceleración
1	1.000
2	1.195
3	1.481
4	2.862
5	2.289
6	4.487
7	5.300
8	7.486

Tabla 6.2 Aceleración Obtenida para el Algoritmo de Floyd-Warshall.

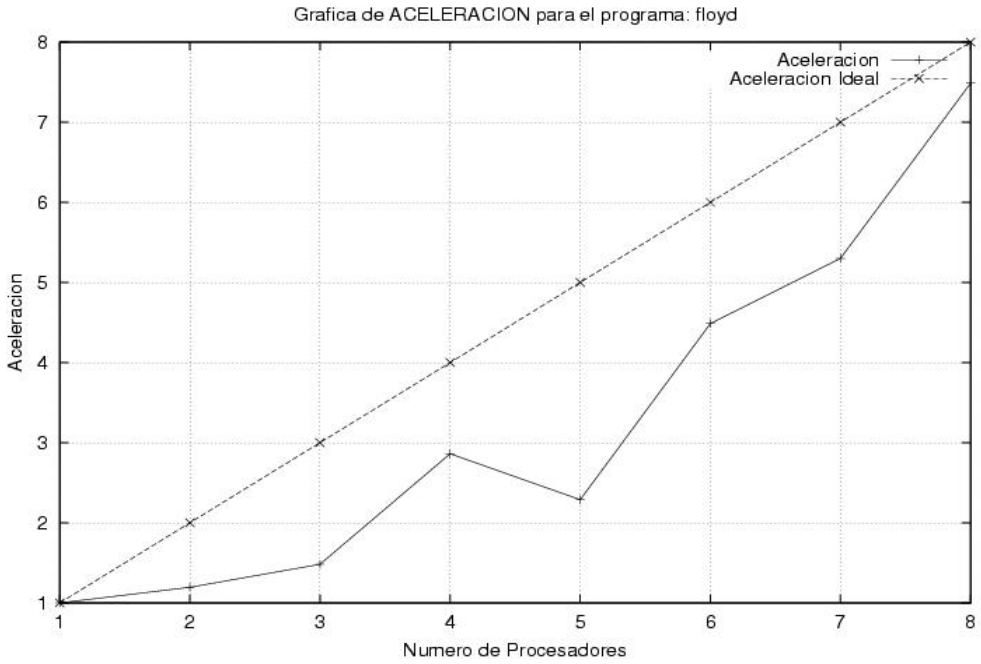


Figura 6.5 Gráfica de Aceleración para el Algoritmo de Floyd-Warshall.

Como puede observarse en los resultados presentados, el algoritmo paralelo implementado para el algoritmo de Floyd-Warshall ofrece un buen desempeño. El tiempo de ejecución va disminuyendo de forma significativa conforme se incrementa el número de procesadores utilizados. Los datos de aceleración indican que una verdadera mejoría de desempeño se obtiene a partir de utilizar 5 ó más procesadores.

6.2 Algoritmos de Jacobi y Gauss

para la resolución sistemas de ecuaciones lineales.

Un ecuación lineal con n variables $x_1, x_2, \ldots, x_n$ es una ecuación la cual puede expresarse como:

$$a_1x_1 + a_2x_2 + \ldots a_nx_n = b$$

donde $a_1, a_2, \ldots, a_n$, y b son constantes.

Un sistema de n ecuaciones lineales con n variables

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots a_{2n}x_n &= b_2 \\ \ldots \quad \ldots \quad \ldots \quad \ldots & \\ a_{n1}x_1 + a_{n2}x_2 + \ldots a_{nn}x_n &= b_n \end{aligned}$$

es usualmente expresado como Ax = b, donde **A** es una matriz de $n \times n$ que contiene los $a_{i,j}$ coeficientes, y **x** y **b** son vectores de n-elementos que contienen las x_i variables y b_i constantes respectivamente.

La ubicación y valor de los elementos diferente de cero de la matriz **A** determina cuán difícil es resolver para **x**. En el caso más general, un algoritmo secuencial que tenga una complejidad $O(n^3)$ puede resolver un sistema lineal de ecuaciones. Dos de los algoritmos más utilizados para la resolución de sistemas lineales de ecuaciones son el de Jacobi y el de Gauss, los cuales se describen brevemente en las siguientes subsecciones.

6.2.1 Método de Jacobi

El método de Jacobi es un algoritmo iterativo para la resolución de sistemas de ecuaciones lineales. Un algoritmo iterativo no da como resultado alguna solución en un número finito de pasos, se puede finalizar su ejecución después de un número finito de iteraciones cuando ha producido una respuesta la suficientemente aproximada a la esperada.

El algoritmo de Jacobi asume que se desea resolver un sistema lineal de ecuaciones **Ax = b** para determinar el valor de un vector **x** desconocido. Note que

$$x_i = \frac{1}{a_{i,i}}\left(b_i - \sum_{j \neq i} a_{i,j} x_j\right)$$

Si conocemos cada valor de x_j donde $j?i$, podemos computar x_i directamente. Pero seguramente, no conocemos estos valores – son precisamente los valores que estamos tratando de determinar – pero si se tiene una estimación para cada un de los valores de x_j, se puede hacer una estimación para x_i[17].

El algoritmo de Jacobi recae en la estimación de cada elemento del vector **x** que da lugar a una nueva estimación para **x**. Utiliza valores computados para cada variable x_i durante la iteración t para generar nuevos valores durante la iteración $t + 1$:

$$x_i(t+1) = \frac{1}{a_{i,i}}\left(b_i - \sum_{j \neq i} a_{i,j} x_j(t)\right)$$

La descomposición de los datos determina donde en el algoritmo es necesaria la comunicación entre procesadores para poder paralelizar el algoritmo. Suponga que cada procesador es responsable por un conjunto contiguo de filas de la matriz **A**, así como también los correspondientes elementos de **b**.

Con esta descomposición de datos, la interacción entre procesos ocurre en dos lugares durante cada iteración del ciclo *do...while*. Primero, cada procesador debe tener una copia del vector **x** computado durante las iteraciones previas. Así cada procesador debe comunicar a todos los demás procesadores (broadcast) sus elementos de **x**.

Segundo, aunque cada procesador puede computar un valor local de *diff* basado en el cambio de valores de sus propios elementos de **x**, estos valores locales deben combinarse para obtener un valor global de *diff*. Así una operación de reducción es necesaria. Al final del paso de reducción, cada procesador tendrá el mismo valor para *diff*, lo que causará que todos los procesadores salgan del ciclo *do…while* en la misma iteración.

```
E ← Criterio de convergencia
Diff ← Diferencia con respecto al criterio de convergencia

Entrada: sistema lineal de ecuaciones
Enviar el arreglo de variables a todos los procesadores
Dividir y enviar la parte correspondiente de la matriz de coeficientes a cada procesador
Dividir y enviar la parte correspondiente del vector de resultados a cada procesador
Refinar los valores estimados de x hasta que los valores converjan, si es que convergen
  Do
  {
    Determinar la diferencia máxima global
    Juntar el vector de variables
    Juntar y reiniciar el vector de variables
  }while(Diff > E)
```

Figura 6.6 Pseudocódigo paralelo del algoritmo de Jacobi.

6.2.1.1 Resultados Obtenidos

Los siguientes resultados fueron obtenidos al ejecutar la aplicación paralela que implementa el método de Jacobi la cual representa un sistema de ecuaciones lineales de 1000x1000, utilizando 8 procesadores para su procesamiento.

Número de Procesadores	Tiempo (segundos)
1	3.900
2	2.226
3	2.436
4	1.046
5	1.428
6	1.678
7	1.489
8	0.763

Tabla 6.3 Resultados Obtenidos para el Algoritmo de Jacobi

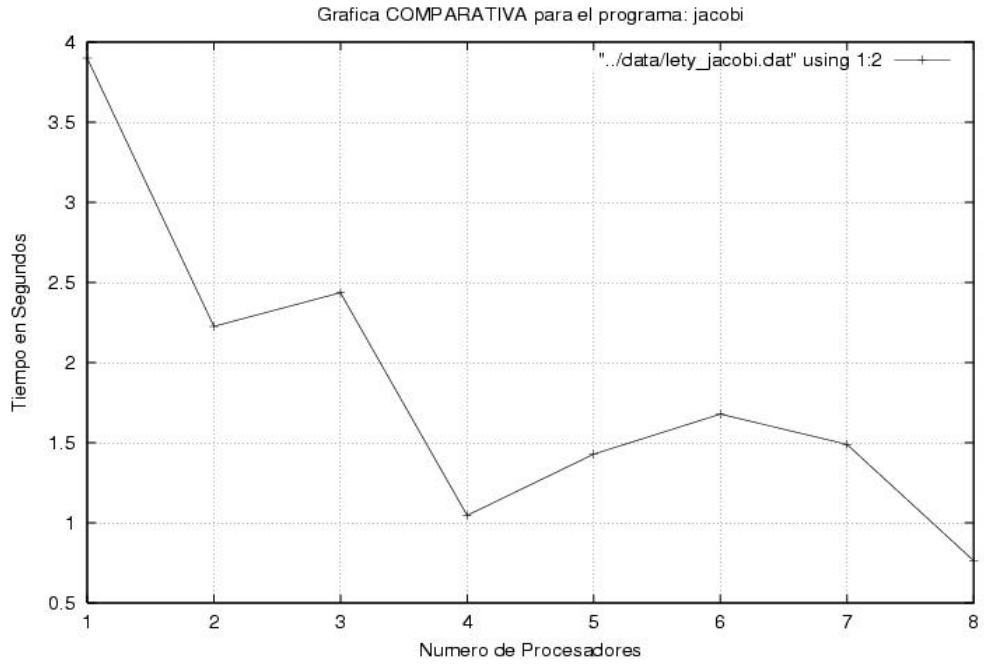


Figura 6.7 Gráfica de Resultados para el Algoritmo de Jacobi.

Número de Procesadores	Aceleración
1	1.000
2	1.752
3	1.601
4	3.728
5	2.731
6	2.324
7	2.619
8	5.111

Tabla 6.4 Aceleración Obtenida para el Algoritmo de Jacobi.

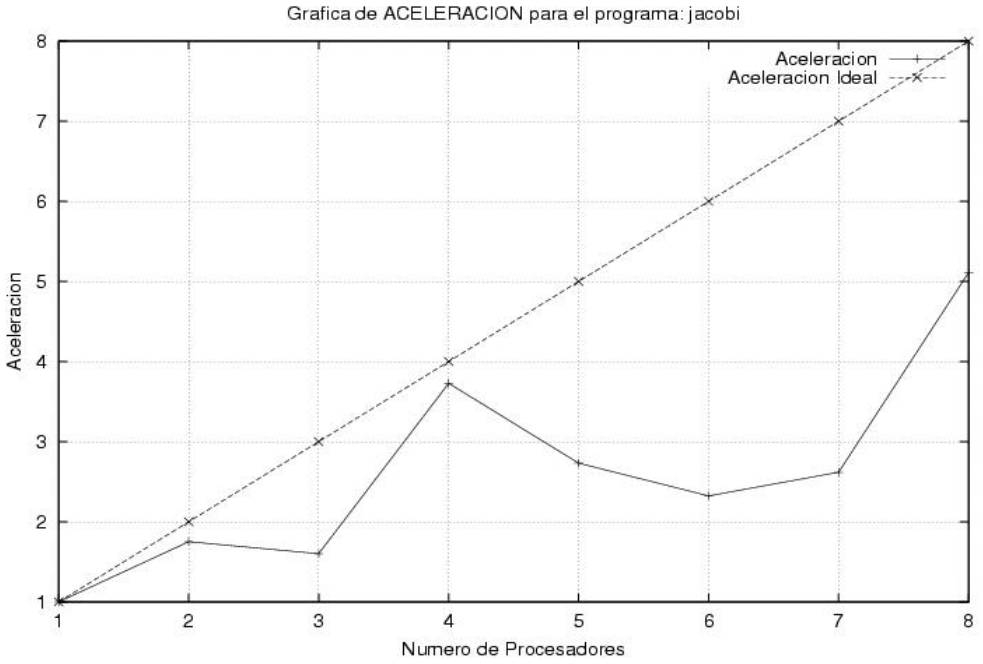


Figura 6.8 Gráfica de Aceleración para el Algoritmo de Jacobi.

6.2.2 Eliminación Gaussiana

En el método de eliminación Gaussiana, operaciones elementales sobre filas son aplicadas en un orden específico para transformar a una matriz aumentada en una triangular escalonada tan eficientemente como sea posible.

La esencia del método es la siguiente: Dado un sistema de *m* ecuaciones con *n* variables, tomar la primera ecuación y sustraer múltiplos adecuados de ella a partir de las *m-1* ecuaciones restantes. En cada caso escoger el múltiplo de tal manera que la sustracción cancele o elimine la misma variable, digamos x_1. El resultado son las restantes *m-1* ecuaciones las cuales solamente contienen *n-1* variables (x_1 no aparece nunca más). Hacer un lado la primera ecuación y repetir el mismo procedimiento con las restantes *m-1* ecuaciones.

El proceso continúa hasta que se cumplan algunas de las siguientes condiciones:

- Solo queda una ecuación con una variable. En tal caso, existe una solución única y una sustitución regresiva se utiliza para encontrar los valores de las demás variables.
- Solo quedan variables pero no ecuaciones. En tal caso no existe una solución única.
- Solo quedan ecuaciones pero no variables (i.e. la columna más baja de la matriz aumentada contiene solo ceros del lado derecho de la línea vertical). Esto indica que el sistema es inconsistente o redundante. En el caso de inconsistencia la información contenida en las ecuaciones es contradictoria. En el caso de redundancia, puede que exista una solución única y una sustitución regresiva pueda utilizarse para encontrar los valores de las otras variables.

Un estudio de la dependencia de datos del algoritmo revela que tanto el ciclo *for* más profundo indexado por *k* y el ciclo *for* de en medio indexado por *j* pueden ejecutarse en paralelo. En otras palabras, una vez que la fila pivote ha sido encontrada, las modificaciones a todas las filas no marcadas pueden ocurrir simultáneamente. Dentro de cada fila, una vez que el multiplicador *a[j][i]/a[picked][i]* ha sido computado, las modificaciones a los elementos *i+1* a través de *n* para cada fila puede ocurrir simultáneamente.

Consideremos la implementación de la eliminación gaussiana en una multicomputadora. Asumimos que *n* es un múltiplo de *p*. Si se examina el flujo de datos del algoritmo por iteración, se observa que la determinación de la fila pivote *picked* requiere que los datos en la columna *i* sean comparados, mientras se determina el nuevo valor de un elemento en particular *a[j][k]* requiere tres valores además del valor actual de *a[j][k]*. Estos valores son *a[j][i]*, *a[picked][i]*, y *a[picked][k]*.

Claramente la distribución de datos determina los puntos en el algoritmo donde las comunicaciones son necesarias. Supóngase que se le asigna a cada procesador un grupo contiguo de filas de *a* y los elementos asociados de *b*. Dada esta distribución de datos, los procesadores deben interactuar en orden para determinar la fila pivote. Una vez que la fila pivote ha sido determinada, el procesador que la posee debe comunicar los elementos de esta a todos los demás procesadores, de tal forma que puedan actualizar los valores de las filas no marcadas que poseen[17]. La Figura 6.9 muestra el pseudocódigo paralelo del algoritmo de Gauss.

```
GAUSSIANELIMINATION( Multicomputadora )
Local n {Tamaño del sistema lineal}
a[1..n/p][1..n] {Matriz de coeficientes}
b[1..n/p] {Vector de coeficientes independientes}
marked[1..n/p] {Arreglo para marcar los renglones}
pivot[1..n/p] {No. de iteración en la cual cada rengón se usó como pivote}
picked {Renglón seleccionado como pivote}
winner {Procesador que determina el pivote}
i, j
begin
  forall P i where 1 ϒ⇐id ϒ⇐p do begin
    for i := 1 to n/p do marked[i] = 0;
  for i := 1 to n-1 do begin
    temp := 0;
  for j := i to n/p do
    if marked[j] == 0 and |a[j][i]| > temp then begin
      temp := |a[j][i]|; picked := j
end;
winner := id;

MAX.TOURNAMENT( id, temp, winner );
GAUSSIANELIMINATION( Multicomputadora )
begin
  forall P i where 1 ϒ⇐id ? ? p do begin . . .
    for i := 1 to n-1 do begin
      if id == winner then begin
        marked[picked] := 1; pivot[picked] := i;
    for j := i+1 to n do tmp[j] := a[picked][j];
      tmp[n+1] := b[picked]
    end;

BROADCAST( id, winner, tmp );
for j:= 1 to n/p do
  if marked[j] = 0 then begin
    temp := a[j][i] / tmp[i]
  for k := i+1 to n do a[j][k] := a[j][k] - tmp[k] *temp;
   b[j] := b[j] - tmp[n+1]*temp
 end
end;
for i := 1 to n/p do
  if marked[j] == 0 then pivot[i] := n
end
```

Figura 6.9 Pseudocódigo paralelo del algoritmo de Gauss

6.2.2.1 Resultados Obtenidos

Los siguientes resultados fueron obtenidos al ejecutar la aplicación paralela que implementa el método de Gauss la cual representa un sistema de ecuaciones lineales de 1000x1000, utilizando 8 procesadores para su procesamiento.

Número de Procesadores	Tiempo (segundos)
1	200.856
2	319.616
3	185.937
4	89.428
5	112.817
6	77.625
7	69.556
8	49.941

Tabla 6.5 Resultados Obtenidos para el Algoritmo de Gauss

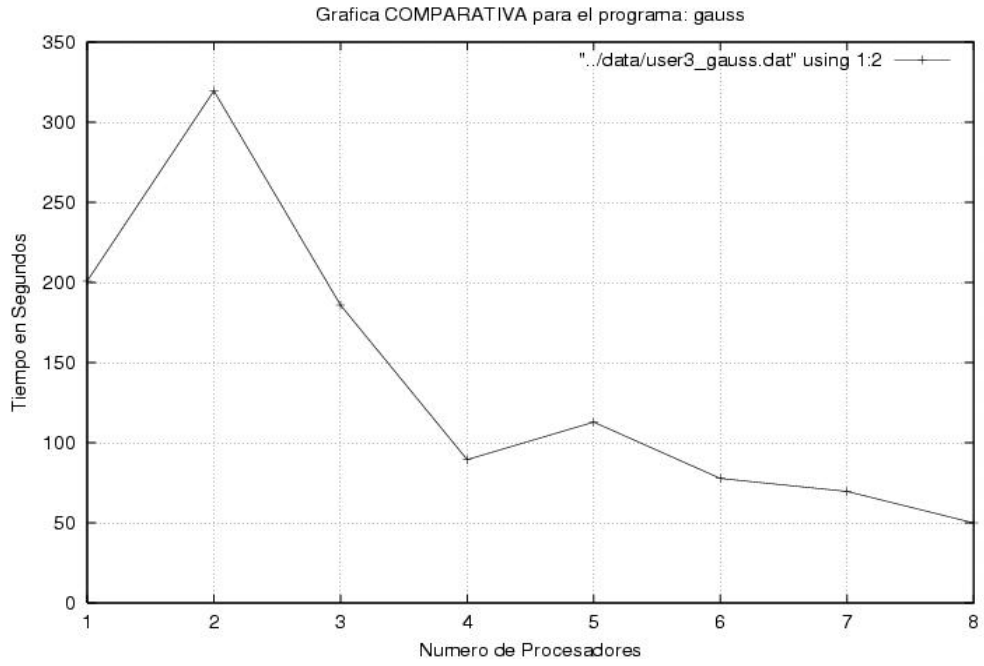


Figura 6.10 Gráfica de Resultados para el Algoritmo de Gauss.

Número de Procesadores	Aceleración
1	1.000
2	0.628
3	1.080
4	2.246
5	1.780
6	2.588
7	2.888
8	4.022

Tabla 6.6 Aceleración Obtenida para el Algoritmo de Gauss.

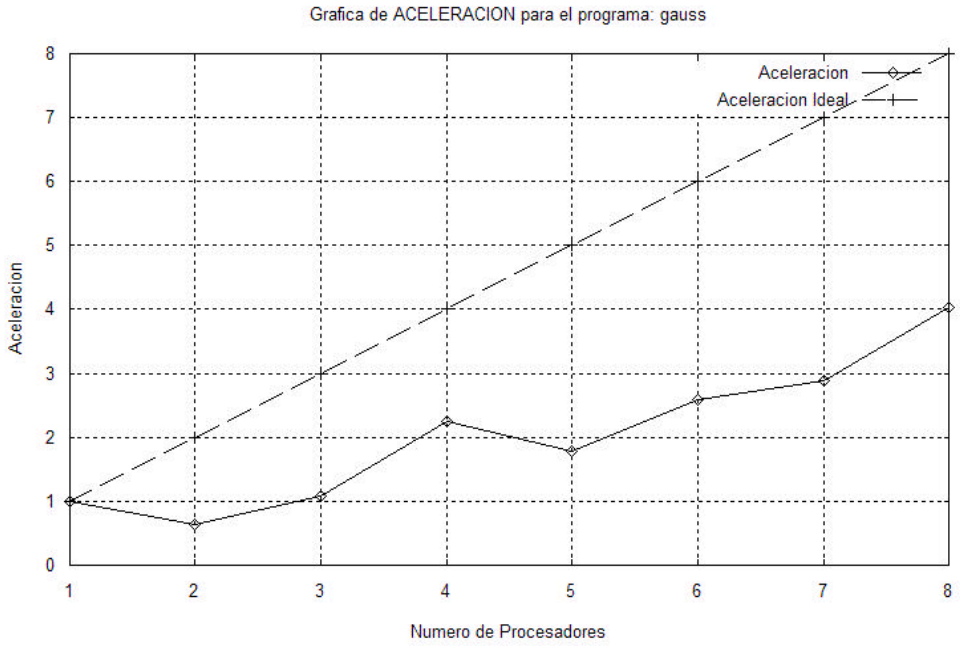


Figura 6.11 Gráfica de Aceleración para el Algoritmo de Gauss.

Los resultados muestran que a pesar de que en general el tiempo de ejecución de la aplicación disminuye conforme se utilizaron más procesadores solamente se consiguió obtener un mejor desempeño relativo al utilizar todos los procesadores disponibles (ocho) y que en instancias anteriores el aumento de desempeño es muy pobre.

Conclusiones

Esta tesis surgió de la necesidad de poder acceder de manera remota las aplicaciones de un sistema, en particular, las aplicaciones paralelas del Laboratorio de Paralelismo de la Sección de Computación del Departamento de Ingeniería Eléctrica. Dentro de los objetivos que se plantearon para satisfacerla, se encuentran:

1. Que el sistema a implementar tuviera una interfaz amigable al usuario, fácil de utilizar y confiable.
2. Que los usuarios pudieran tener acceso a los recursos del sistema desde su lugar de origen, evitando la necesidad de trasladarse o de utilizar algún software especial.
3. Que el sistema mostrara al usuario la información necesaria para que este pudiera estar pendiente de la ejecución del recurso ejecutable seleccionado, mostrándole gráficamente al terminar su ejecución los resultados obtenidos.
4. Que los recursos con los que cuenta el Laboratorio de Paralelismo fueran aprovechados de la mejor forma posible, al crear una herramienta de calendarización.

Los primeros tres puntos, que en general, conforman la interfaz entre el usuario y el sistema con la cual interactúa para poder acceder a las aplicaciones, fueron logrados utilizando tecnologías como *scripts CGI en lenguaje C*, *HTML*, y *Java (Java Swing)*, y herramientas de graficación como *gnuplot*. Estas tecnologías permitieron crear una interfaz que fuera independiente del sistema en que se ejecutara y que pueda utilizarse desde el lugar de origen del usuario.

Para lograr el último punto, se construyó una Herramienta de Calendarización la cual optimiza la forma en la cual los recursos del Laboratorio de Paralelismo son utilizados (particularmente los ocho procesadores con los que cuenta), para satisfacer las diversas solicitudes enviadas por los usuarios. Tecnologías como el *lenguaje C*, *Threads*, y *Sockets* hicieron posible la implementación de esta herramienta, la cual consta de un *Despachador de Corto y Mediano Plazo*. El despachador de mediano plazo esta continuamente "escuchando" por peticiones de usuario, las cuales agrega a una lista de tareas atendida periódicamente por el despachador de corto plazo para le ejecución de dichas peticiones.

En el caso particular del punto tres, se creo una *Herramienta de Visualización* utilizando el lenguaje Java (Java Swing y temporizadores), la cual muestra gráficamente de forma porcentual el avance de cada uno de los procesadores involucrados en la ejecución de la aplicación seleccionada por el usuario, así como también muestra las diversas actividades realizadas por los despachadores de corto y mediano plazo. Esto sin embargo es realizado de manera asíncrona con respecto a la ejecución de la aplicación seleccionada, lo cual representa una solución parcial ya que lo ideal sería una relación 1:1 entre la herramienta de visualización y la ejecución de la aplicación o al menos que solo existiera una diferencia muy pequeña, aunque hay que tomar en cuenta que la información que utiliza la herramienta de visualización tiene que "viajar" desde el Laboratorio de Paralelismo hasta la máquina del usuario, lo cual puede representar un gran obstáculo.

El sistema presentado en esta tesis demuestra que es posible aprovechar de una mejor manera las instalaciones con las que cuente una institución (como el Laboratorio de Paralelismo) haciendo uso de las nuevas tecnologías para la creación de páginas Web dinámicas y aplicaciones Web. Puede servir como base o punto de referencia para la implementación de un sistema más robusto, agregando características como: (i) Permitir al usuario subir al servidor su propia aplicación (la cual tendría que contar con ciertas características, como la de escribir información para la herramienta de visualización) y ejecutarla, y (ii) expandir los tipos de gráficas de resultados.

Referencias

[1] A.A. Khan, C. L. McCreary and M. S. Jones, **A comparison of multiprocessor scheduling heuristics**, In Int'l Conf. on Parallel Processing volume II, pages 243-250, August 1994

[2] Chow Randy, Jonson Theodore, **Distributed Operating Systems & Algorithms**, Addison-Wesley, 1997

[3] Coffman, E., Jr., and R. Graham. 1972. **Optimal scheduling for two processor systems**, Acta Informática, vol. 1, pp. 200-213

[4] Copeland R. Dennos, Corbo C. Raymond, Falkenthal A. Susan, Fisher L. James and Sandler N. Mark, **Which Web Development Tool Is Right for You?**, IT Professional, March-April 2000, Volume 2 2, Page(s): 20-27

[5] Cormen, T. H., Leiserson, C. E., and Rivest R. L., **Introduction to Algorithms,** MIT Press/McGraw-Hill, 1990

[6] Culler David, Singh J. P., Gupta Annop, **Parallel Computer Architecture: A Hardware/Sofware Approach**, Morgan-Kaufmann Publishers, August 1998

[7] D. Ragget, A. Le Hors, I. Jacobs, **HTML 4 Specification**, 1998.

[8] Daoud Fawzi, **Electronic Commerce Infrastructure**, IEEE Potentials, February/March 2000, Page(s): 30-33

[9] Dongarra J. Jack, Otto W. Steve, Snir Marc, Walter David, **A Message Passing Standard for MPP and Workstations**, Communications of the ACM, July 1996, Vol. 39, No. 7, Page(s): 84-90

[10] Eckel Bruce, **Thinking in Java**, Second Edition, Prentice-Hall.

[11] Ferrero Alessandro, Piura Vicenzo, **A Simulation Tool for Virtual Laboratory Experiments in a WWW Enviroment**, IEEE Instrumentation and Measurement, Technology Conference, St. Paul, Minnesota, USA, May 18-21 1998.

[12] Foster Ian, **Designing and Building Parallel Programs**, Addison-Wesley, 1995

[13] G. C. Sih and E. A. Lee, **A compile time scheduling heuristic for interconnection constrained heterogeneous processor architectures**, IEEE Trans Parallel and Distributed System, 4(2):175-187, February 1993

[14] Gundavaram Shishir, **CGI Programming on the World Wide Web**, 1st Editon, March 1996, O'Reilly

[15] H. El Rewini and T. G. Lewis, **Scheduling parallel program tasks onto arbitrary target machines**, Journal of Parallel and Distributed Computing, June 1990

[16] Lewis Ted, El Rewini Hesham, **Parallax: A tool for parallel program scheduling**, IEEE Parallel and Distributed Technology, pp 62-72, 1993

[17] M.J. Quinn, **Parallel Computing: Theory and Practice**, McGraw-Hill, 1994

[18] M. R. Gary and D. S. Johnson, **Computers and Intractability A Guide to the Theory of NP Completeness**, W H Freeman and Company, 1979

[19] M. Y. Wu and D. D. Gajski, **Hypertool A programming aid for message passing systems**, IEEE Trans Parallel and Distributed Systems, 1(3):330-343, July 1990

[20] Matt J. Crouch, **Web Programming with ASP and COM**, Ed. ADDISON-WESLEY, Diciembre 1999

[21] McCray T. Alexa, Gallagher E. Marie, **Principles For Digital Library Development**, Communications of the ACM, May 2000, Vol 44. No. 5

[22] Middelton Hill, Deng Brian, Kemp Chris, Web **Programming With Perl5**, March 1997, SAMS

[23] Murray Jeanne, **Building Java HTTP Servlets**, IBM developerWorks

[24] North Ken, **DBMS ODBC Special Report**, April 1996

[25] Pressman S. Roger, Ingeniería del Software, **Un Enfoque Práctico**, Cuarta Edición, McGraw-Hill

[26] Ridge Daniel, Becker Donald, Merkey Phillip, Sterling Thomas, **Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs**, Proceedings, IEEE Aerospace, 1997

[27] Schlueter Christoph, Shaw J. Michael, **A Strategic Framework For Developing Electronic Commerce**, IEEE Internet Computing, November-December, Vol. 1, No. 6, Page(s): 20-28

[28] Sterling Thomas, Becker J. Donald, Savarese Daniel, Dorband E. John, Ranawak A. Udaya, Packer V. Charles, **Beowulf: A Parallel Workstation For Scientific Computation**, Proceedings of the 24th International Conference on Parallel Processing (ICPP), August 1995, Page(s):11-14

[29] T. Yang and A. Gerasoulis, **DSC Scheduling parallel tasks on an unbounded number of processors**, IEEE Trans Parallel and Distributed System, 5(9):951-967, September 1994

[30] Ullman J.D., 1975, **NP-complete scheduling problems**, Journal of Computer and System Sciences vol. 10, pp. 384-393

[31] V. Sarkar, **Partitioning and Scheduling Parallel Programs for Multiprocessors**, The MIT Press, 1989

[32] Y. C. Chung and S. Ranka, **Applications and performance analysis of a compile time optimization approach for list scheduling algorithms on distributed memory multiprocessors**, In Supercomputer '92, November 1992

[33] Yu-Kwong Kwok, Ishfaq Ahmad, **Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors**, ACM Computing Surveys, 1998

[34] http://www.cpan.org/