UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
CAMPUS ARAGÓN

"DESARROLLO DE ÉXIGUS, UNA HERRAMIENTA QUE PERMITE IMPLEMENTAR CÓMPUTO DISTRIBUIDO EN PROYECTOS"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN

P R E S E N T A:

DAVID GUERRERO GUERRA

ASESOR DE TESIS:
M. EN C. MARCELO PÉREZ MEDEL

MÉXICO, 2004.

A mis padres, Elena y Guillermo

A toda la comunidad de verdaderos hackers del mundo,
que ponen su trabajo y esfuerzo a disposición de todos,
de forma desinteresada.

Agradecimientos

Antes que nada, mil gracias Dios, por darme la oportunidad y el ánimo suficiente para terminar todo este trabajo que simplemente no tenía ni pies ni cabeza. Y aunque no es lo único por lo que te estoy agradecido, es algo que no quería que se me pasara.

Muchas gracias a mis revisores: el M. en C. Jesús Díaz, el M. en C. Leobardo Hernández, el Ing. Alejandro René González y el Ing. Rodolfo Vázquez por sus valiosos comentarios y correcciones con la finalidad de realizar un trabajo de calidad. Pero en especial, gracias a mi asesor, el M. en C. Marcelo Pérez, que aunque lo conocí todavía siendo Ingeniero, nunca ha perdido la sencillez y la disposición para transmitir sus conocimientos, experiencia y ánimo, no sólo a sus alumnos, sino aquellos que han tenido oportunidad de conocerlo.

Gracias a todos mis profesores de la carrera, de los cuales aprendí muchísimas cosas valiosas, no solamente relacionadas con la computación, sino con otras áreas. Gracias también a todos mis compañeros de clases, los cuales hicieron muy agradables todos los años de la carrera y que aunque aquí no ponga todos sus nombres (principalmente porque sólo conozco sus apodos ;), los recuerdo a cada uno de ustedes. Gracias Alicia, Abel, Felipe, Yossiane, Octavio, Raúl, Rafael, Luis, Sócrates, Daniel, Aarón, Serafín y todos los que me faltaron.

También quiero agradecer a todas las personas del Laboratorio de Cómputo del Centro Tecnológico Aragón, tanto por su apoyo técnico como moral, en especial a Adriana, Flor, Lluvia, Arturo, Ernesto, Jesús y Rodrigo.

Sin embargo, como no todo en la vida es trabajo y estudio, hay muchísima más gente no involucrada en estas actividades a las cuales también les tengo algo que agradecer.

En primer lugar, muchísimas gracias a todo mi familia, a mis padres Elena y Guillermo, mi abuela Sara, mis hermanos Diana, César y Guillermo. No me alcanzaría el tiempo y el espacio para detallar todo por lo que les estoy agradecido, así que simplemente, muchas gracias.

Muchas gracias a todos mis viejos amigos, los cuales los conozco durante varios años, en algunos casos, llevo media vida de conocerlos. Muchas gracias a Abraham, Ricardo, Gerardo y David V.

Muchas gracias a mis nuevos amigos, que aunque no los conozco de tanto tiempo, francamente no siento ninguna diferencia. Gracias Elena, Arelí, Fer, Fernando, Sergio, Christian y Juan.

Gracias especiales al Ing. Jorge Rodríguez. En primer lugar, por ser un gran proveedor de material digital. Y en segundo lugar, por ser una persona generosa con la que se puede confiar y contar con su apoyo. ¡Gracias brother!

De nueva cuenta, gracias a todos y una disculpa a los que me faltaron. Pero de forma súper especial, gracias Dios, por esta vida maravillosa.

If it was hard to write, it should be hard to understand.

–A Unix programmer

INTRODUCCIÓN

En 1988 Arjen K. Lenstra y Mark S. Manasse del DEC System Research Center escriben un software para factorizar números grandes. Este software tomaba el número que se deseaba factorizar y en base a él se creaban varios límites de divisores sobre los cuales trabajar. Por medio de correo electrónico estos límites se distribuían entre estaciones de trabajo tanto dentro como fuera del laboratorio, de tal forma que cada computadora trabajaba con sólo una parte del problema. Una vez que se obtenían los resultados, estos se regresaban al laboratorio utilizando el mismo correo electrónico. Esta técnica tuvo un resultado sorprendente. En 1990, con la colaboración de más de 100 computadoras alrededor del mundo fue posible factorizar un número de cerca de 100 dígitos decimales. Tres años después, con la colaboración de más de 600 computadoras se realizó la factorización de un número de 129 dígitos, con lo que el grupo se hizo acreedor a un premio monetario otorgado por RSA Security

Hoy en día existen proyectos de computo distribuido similares al descrito anteriormente, sin embargo, su finalidad es muy distinta a la de factorizar números. Sus objetivos van desde buscar evidencia de inteligencia extraterrestre hasta la investigación y desarrollo de medicamentos nuevos. Utilizando como infraestructura de comunicaciones Internet, ha sido posible que gente común y corriente colabore con un poco del poder de procesamiento sin utilizar de sus computadoras para lograr un fin común.

El éxito que han tenido esta clase de proyectos tal vez motive a investigadores, programadores, instituciones, etc. a crear uno similar, sin embargo, pronto se darán cuenta

que existen pocas herramientas específicas para su creación y la complejidad de las existentes puede ser un factor desalentador.

El objetivo de esta tesis es la creación de herramientas que faciliten la creación de proyectos de cómputo distribuido basados en Internet, ofreciendo flexibilidad, escalabilidad y seguridad. Este conjunto de herramientas se denominará Éxigus y se pondrá a disposición del público por medio de la licencia GPL.

El trabajo en esta tesis está organizado de la siguiente forma. En el capítulo 1 se muestra una breve introducción al complejo tema del cómputo distribuido y se muestran algunos de los proyectos existentes más importantes en cuestión al número de colaboradores. Se hace énfasis especial en aquellos aspectos que influyen de forma directa con el objetivo de la tesis.

En el capítulo 2 se hace un análisis de algunas de herramientas existentes para crear proyectos de cómputo distribuido. Todos los análisis se hacen bajo 6 aspectos que cubren los detalles más importantes de cada uno.

En el capítulo 3 se muestra el diseño y la implementación del conjunto de herramientas Éxigus de tal forma que se ofrezca flexibilidad, escalabilidad, seguridad y facilidad de uso con una orientación clara a una infraestructura de comunicaciones como es Internet.

En el capítulo 4 se realizan una serie de pruebas de Éxigus, desde su instalación hasta la conversión de un par de aplicaciones típicas a su equivalente en cómputo distribuido.

Finalmente en el capítulo 5 se muestran las conclusiones del trabajo realizado y una serie de recomendaciones tanto para los que quieran extender esta herramienta o los que deseen crear una nueva tomando como base ésta.

Contenido

INTRODUCCIÓN 5
1. INTRODUCCIÓN AL CÓMPUTO DISTRIBUIDO 10
1.1 DEFINICIÓN 11
1.1.1 Clases de cómputo distribuido 13
1.2 HISTORIA 16
1.3 ASPECTOS DE DISEÑO 18
1.3.1 Modelos de programación 19
1.3.2 Planificación 20
1.3.4 Seguridad 21
1.3.5 Tolerancia a fallos 22
1.4 EJEMPLOS DE PROYECTOS DE CÓMPUTO DISTRIBUIDO 23
1.4.1 SETI@home 24
1.4.2 Distributed.net 27
1.4.3 GIMPS 30
1.4.4 Otros proyectos 32
1.5 CONCLUSIONES 34
2. HERRAMIENTAS PARA CREAR PROYECTOS DE CÓMPUTO DISTRIBUIDO 35
2.1 ACERCA DE LOS ANÁLISIS REALIZADOS 36
2.2 FIDA 38
2.2.1 Información técnica 38
2.2.2 Arquitectura 39
2.2.3 Características especiales 40
2.2.4 Seguridad y tolerancia a fallos 41
2.2.5 Creación de una aplicación 43
2.2.6 Evaluación 43
2.3 MODELS@home 44
2.3.1 Información técnica 45
2.3.2 Arquitectura 45
2.3.3 Características especiales 47
2.3.4 Seguridad y tolerancia a fallos 47

2.3.5 Creación de una aplicación 48
2.3.6 Evaluación 49
2.4 Q²ADPZ 50
2.4.1 Información técnica 50
2.4.2 Arquitectura 50
2.4.3 Características especiales 53
2.4.4 Seguridad y tolerancia a fallos 54
2.4.5 Creación de una aplicación 55
2.4.6 Evaluación 56
2.5 BOINC 57
2.5.1 Información técnica 58
2.5.2 Arquitectura 58
2.5.3 Características especiales 60
2.5.4 Seguridad y tolerancia a fallos 61
2.5.5 Creación de una aplicación 62
2.5.6 Evaluación 64
2.6 CONCLUSIONES 65
3. DISEÑO E IMPLEMENTACIÓN DE ÉXIGUS 66
3.1 OBJETIVOS DE LA INFRAESTRUCTURA ÉXIGUS 66
3.2 CONSIDERACIONES PARA EL DISEÑO DE LA ARQUITECTURA 68
3.2.1 Requisitos para realizar un trabajo 69
3.2.2 Aspectos a considerar en los trabajos 71
3.2.3 Aspectos a considerar para los resultados 73
3.2.4 Solicitud y asignación de trabajos 74
3.2.5 Categorización y control de los trabajos 75
3.2.6 Tolerancia a fallos en los resultados 77
3.2.7 Protección del trabajador y los datos en tránsito 83
3.2.8 Protección del maestro contra ataques de negación de servicio 85
3.2.9 Balanceo de la carga 86
3.3 DISEÑO DE LA ARQUITECTURA 88
3.3.1 Datos en tránsito 89
3.3.2 Diseño del trabajador 98
3.3.3 Diseño del maestro 108
3.4 IMPLEMENTACIÓN DEL TRABAJADOR 116

3.4.1 Archivos y estructura de directorios necesaria 117
3.4.2 Descripción del flujo de ejecución 119
3.5 IMPLEMENTACIÓN DEL MAESTRO 123
3.5.1 Implementación de la base de datos 123
3.5.2 Implementación del servidor Web 127
3.5.3 Implementación del Coordinador 130
3.6 CONCLUSIONES 135
4. PRUEBAS 136
4.1 PROYECTO ESPERA 137
4.1.1 Introducción 137
4.1.2 Descripción de los trabajos, aplicaciones y herramientas creadas 137
4.1.3 Pruebas realizadas en uno y varios equipos físicos 138
4.1.4 Interpretación de los resultados 141
4.2 PROYECTO TRAZADO DE RAYOS 143
4.2.1 Introducción 143
4.2.2 Descripción de los trabajos, aplicaciones y herramientas creadas 144
4.2.3 Pruebas realizadas en uno y varios equipos físicos 146
4.2.4 Interpretación de los resultados 148
4.3 PROYECTO FACTORIZACIÓN 148
4.3.1 Introducción 148
4.3.2 Descripción de los trabajos, aplicaciones y herramientas creadas 151
4.3.3 Pruebas realizadas en uno y varios equipos físicos 153
4.3.4 Interpretación de los resultados 155
4.4 CONCLUSIONES 156
5. RECOMENDACIONES Y CONCLUSIONES 157
5.1 RECOMENDACIONES 157
5.2 CONCLUSIONES 160
REFERENCIAS 163
GLOSARIO 165
ANEXO A Preparación del entorno de ejecución del Maestro bajo plataforma Windows 166
ANEXO B Uso del ambiente de pruebas contenido en el CD 169
ANEXO C. Guía práctica para el uso de la infraestructura Éxigus 185

1. INTRODUCCIÓN AL CÓMPUTO DISTRIBUIDO

A finales de los años 90 y principios del 2000 el término cómputo distribuido comienza a hacerse popular entre usuarios comunes de PCs[1] e Internet debido a la apertura de páginas Web de proyectos como SETI@home [30], distributed.net [9] y GIMPS [37]. Como se ha estado haciendo desde entonces, un usuario de forma voluntaria descarga un software de alguna de estas páginas Web el cual generalmente es un protector de pantalla y lo instala en su computadora. Este software permanece latente en la computadora y sólo mientras exista inactividad por parte del usuario se ejecutará un proceso especialmente diseñado. Este proceso permite que la computadora en donde se ejecute colabore junto con otras cientos, miles o millones alrededor del mundo conectadas a Internet para resolver un problema en particular, el cual puede ser encontrar evidencias de inteligencia extraterrestre (SETI@home), descifrar un código secreto (distributed.net) o buscar números primos grandes (GIMPS).

Debido a esto el término de cómputo distribuido se asoció a una gran colaboración en la cual las personas participan donando tiempo sin usar de los CPUs[2] de sus computadoras por medio de un software especializado e Internet [5]. Aunque este concepto no es totalmente

---

[1] Computadoras Personales

[2] Unidades Centrales de Procesamiento

erróneo, el cómputo distribuido tiene un sentido más amplio, como lo veremos a lo largo de este capítulo.

Desde la creación de las primeras redes de estaciones de trabajo, se han realizado muchos estudios sobre el cómputo distribuido. Este es un tema bastante amplio y de cierta forma, en constante evolución, por lo que solamente se puede presentar la parte fundamental del mismo.

1.1 DEFINICIÓN

Una definición del cómputo distribuido indica que éste es un sistema computacional en el cual varias computadoras interconectadas comparten las labores asignadas al sistema [31]. Sin embargo, ésta no es la única: conjunto de procesos que cooperan para lograr un fin común [1] y distribución de aplicaciones y lógica de negocios entre múltiples plataformas de procesamiento [21] son dos más de las muchas definiciones que existen sobre este tema.

A pesar de la falta de consenso en una definición única se pueden destacar dos aspectos a los cuales concuerda la mayoría:

1.- El cómputo distribuido involucra dos o más computadoras completas (microprocesador, memoria y Entrada/Salida) comunicadas entre sí de alguna forma (generalmente un red de comunicaciones).

2.- Algunas o todas las computadoras involucradas participan de forma secuencial o simultánea con un fin común.

Cabe destacar que estos dos puntos involucran aspectos de diseño y no de implementación. Es decir, en el cómputo distribuido existen una serie de recomendaciones más no imposiciones de cómo éste debe llevarse a cabo, tanto en hardware como software, ya que la implementación depende del fin que se desee lograr.

Tomando en cuenta los conceptos anteriores se pueden presentar dos escenarios a manera de ejemplo en los cuales se lleva a cabo el cómputo distribuido:

1.- Un portal de Internet con varios servidores Web y uno de base de datos. Los servidores Web atienden las solicitudes de navegadores y el servidor de base de datos se encargan de almacenar y ordenar la información que se genere para el momento que se requiera. Esto con el fin de dar un servicio adecuado a los clientes o visitantes del portal.

2.- Una red de estaciones de trabajo para fines de rendering[3]. En este caso, uno o varias computadoras miembros de la red se encargan de asignar a las computadoras restantes un conjunto de escenas que son parte de una animación. Una vez que la representación de las escenas ha concluida se envían las imágenes, ya sea al mismo equipo que las asignó u otro diferente.

Este último escenario en particular es a veces considerado propio del cómputo paralelo, por lo que es conveniente hacer una aclaración. Al igual que el cómputo distribuido no hay un acuerdo total en la definición exacta de cómputo paralelo, sin embargo la mayoría acepta que este último es un conjunto de técnicas y procedimientos para disminuir el tiempo de solución de un problema dividiéndolo en partes y resolviendo cada parte en una unidad de procesamiento de forma simultánea.
Si esto lo comparamos con el segundo aspecto de cómputo distribuido antes mencionado, se observa que "problema" y "fin común" son equivalentes y por lo tanto se puede decir que el cómputo distribuido es inherentemente paralelo o viceversa.

Con el fin de separar el cómputo paralelo del distribuido se ha señalado que en el cómputo paralelo cada unidad de procesamiento ejecuta el mismo código mientras que en el distribuido cada computadora ejecuta código diferente, lo cual es cuestionable. Para fines de esta tesis se utilizará la siguiente condición, aceptada y rechazada por igual: el cómputo paralelo involucra dos o más unidades de procesamiento (microprocesador y memoria, mas no Entrada/Salida) comunicadas entre sí de alguna forma.
Como se puede notar, es el hardware y no el software el que da la pauta para marcar la división entre uno y otro.

---

[3] rendering: proceso de convertir figuras geométricas tridimensionales y fuentes de luz en una representación bidimensional.

1.1.1 Clases de cómputo distribuido

Con el paso del tiempo han surgido especializaciones o clases del cómputo distribuido. Estas clases se caracterizan por propuestas de una ideología de uso e implementación o por definir de forma concreta aspectos como la infraestructura o forma de comunicación de las computadoras involucradas.
La siguiente es una muestra de las diferentes clases que existen. Cabe señalar que debido a su origen común, muchas clases comparten elementos comunes que dificultan su diferenciación.

Metacómputo

El metacómputo es una arquitectura distribuida con la peculiaridad de que su red de comunicación es Internet [20]. Su rasgo más característico es que utiliza supercomputadoras, clusters y estaciones de trabajo, además de PCs, como unidades de procesamiento dentro de su arquitectura distribuida.

Debido a su naturaleza heterogénea, este ambiente se programa utilizando HPF (Fortran de Alto Desempeño) y C junto con librerías de paso de mensajes. También se usan tecnologías de distribución de objetos como Java RMI/Corba/DCOM.

Ya que su red de comunicación no es segura, es necesario establecer un mecanismo de seguridad entre las distintas unidades de procesamiento, también conocidas como nodos.

Cabe aclarar que el metacómputo es una extensión de las Redes de Estaciones de Trabajo o NOW (por las siglas en inglés de Network Of Workstation). La diferencia con las NOW es que éstas utilizan solamente estaciones de trabajo y PCs como unidades de procesamiento, junto con una infraestructura de comunicaciones de tipo LAN.

Cómputo de rejilla o red

Una rejilla computacional es una infraestructura de hardware y software que proporciona capacidades computacionales de alto nivel de forma confiable, consistente, accesible y barata [15].

La idea de las rejillas es facilitar la compartición, selección y adición de una gran variedad de recursos incluyendo supercomputadoras, estaciones de trabajo, PCs, sistemas de almacenamiento, fuentes de datos y dispositivos especializados que están geográficamente distribuidos y que son propiedad de diferentes organizaciones para resolver problemas de gran escala computacional y de datos en ciencias, ingeniería y comercio.

Cómputo voluntario

El cómputo voluntario permite que redes de cómputo de alto desempeño sean formadas de una manera rápida, fácil y de bajo costo por medio de la compartición de ciclos sin usar de las computadoras de usuarios conectadas a Internet sin necesidad de ayuda experta [28]. Esto se logra con Java y navegadores de Internet. Con sólo cargar una página de Internet se pueden descargar applets o pequeños programas que se encargan de colaborar en algún problema global mientras el navegador esté mostrando la página. Cuando se cierra la página, la participación se detiene hasta el nuevo ingreso a la misma.

Computo P2P

La tecnología de Par a Par (P2P) permite el intercambio directo de servicios entre computadoras actuando como iguales en una red [26]. Estos servicios incluyen intercambio de información, ciclos de procesamiento, almacenamiento de archivos. Ejemplos comunes de aplicaciones P2P incluyen: Tableros de boletines de noticias de Usenet (información), transacciones EDI (procesamiento), Internet Relay Chat (comunicación) y Napster (archivos)

Las principales características del cómputo P2P son las siguientes:

1.- En el cómputo P2P, el intercambio de servicio es simétrico, esto es, que ocurre entre computadoras corriendo el mismo software y con capacidades y responsabilidades equivalentes.

2.- Puede haber cualquier número de computadoras de los cuales cualquiera puede iniciar una interacción con otro igual.

El siguiente cuadro muestra un resumen con las características de esta muestra de clases de cómputo distribuido:

	Sistema Metacómputo	**Sistema de rejilla**	**Sistema de cómputo voluntario**	**Sistema P2P**
Unidades de procesamiento (nodos)	Supercomputadoras, Clusters, Workstations, PC's	Supercomputadoras, Clusters, Workstations, PC's	PC's, PDA's	PC's
Número de nodos	100-10000	100-10000	10-100	10-10000
Red de comunicación	Internet	Internet	Internet, intranet	Internet
S.O de los nodos	Heterogéneo	Heterogéneo	Heterogéneo	Típicamente homogéneo
Seguridad entre nodos	Necesaria	Necesaria	No necesaria	Raramente requerida
Protocolos	Abiertos	Abiertos, estándares, propósito general	No definidos	Abiertos y cerrados
Propósito	Integrar sistemas diversos y obtener una capacidad de cómputo como un todo, mayor a la suma de las partes.	Proporcionar acceso a capacidades computacionales de alto nivel de forma confiable, consistente, accesible y barata	Formar redes de cómputo de alto desempeño de manera rápida, fácil y de bajo costo.	Intercambio directo de servicios de todo tipo

Cabe preguntarse cual es la clase de cómputo a la que pertenece SETI@home o distributed.net, si es que hay una. La respuesta no es muy alentadora. Debido al éxito de ambos proyectos se les suele asociar a un gran número de clases de forma indistinta. Sin embargo, existe una clase de cómputo que se acerca bastante a los elementos característicos de los proyectos antes mencionados. Éste se denomina cómputo de alto caudal de procesamiento.

El cómputo de alto caudal de procesamiento se caracteriza en que el proceso de resolver un problema o tarea requiere de muchas operaciones (y bastante tiempo) para que al final los resultados que se obtengan ocupen poco espacio en memoria (generalmente disco duro) [17]. Un ejemplo es un programa que encuentra la clave que descifra un mensaje por medio de prueba y error. Mientras que el proceso de búsqueda puede tomar varias horas, días e incluso meses, la clave encontrada no ocupará más de unas cuantas decenas de bytes.

Sólo falta indicar que el cómputo de alto caudal de procesamiento será uno de los factores principales en el diseño e implementación de las herramientas planteadas como objetivo en esta tesis.

1.2 HISTORIA

Una de las arquitecturas de computadoras más antiguas es también la más usada en la actualidad (implementada en las PCs) y se le conoce como computadora escalar, secuencial o de Von Neumann en honor a su creador. En esta arquitectura el procesador obtiene instrucciones y datos de una memoria, se realizan una serie de operaciones y se escriben los resultados de vuelta en la memoria. Este es un proceso altamente secuencial por lo que la velocidad de ejecución de un programa depende directamente de la velocidad de los componentes que la conforman (en especial el procesador) [5].

A principios de los años 60 los investigadores se dieron cuenta que no era posible incrementar la velocidad de los componentes de la computadora secuencial debido a motivos técnicos o económicos propios de la época, por lo que decidieron crear la arquitectura o computadora paralela con el fin de obtener un mayor desempeño. Una computadora paralela es un sistema hecho de varias unidades de proceso identificables trabajando juntas al mismo tiempo [17]. La computadora paralela se suele dividir en dos tipos: las denominadas multiprocesador y las multicomputadoras.
Una computadora multiprocesador [17] es aquella en la que sus procesadores pueden ejecutar flujos separados de instrucciones, pero tienen acceso a un solo espacio de direcciones (también conocido como memoria compartida); la diferencia con una multicomputadora [17] consiste en que cada procesador tiene su propia memoria privada y no puede acceder directamente a la memoria de otro, por lo que se maneja más de un solo espacio de direcciones (también conocido como memoria separada o distribuida).

La arquitectura paralela resultó una buena alternativa para las carencias de cómputo y salvo por unos detalles, es usada en las supercomputadoras actuales. Sin embargo, su diseño e implementación representan más retos que el de una computadora secuencial, esto aunado a una estructura rígida y costos mayores propició arquitecturas alternativas.

No pasó mucho tiempo para que surgiera la idea de extender la arquitectura paralela de forma que en vez de conectar procesadores se unieran computadoras completas. Sin embargo, la tecnología de la época no la hacía práctico.

La década de los años 70 y principios de los años 80 marcan el inicio del cómputo distribuido. Gracias a los avances tecnológicos fue posible disponer del poder de una mainframe en una computadora de escritorio y unir varias de estas por medio de redes de área local.

Los primeros usos del cómputo distribuido se alejan de los actuales. En un principio su finalidad era repartir el trabajo de tal forma que cada computadora tuviera funciones distintas. Por ejemplo, una computadora se encargaba del administrador de base de datos, otra de mantener un sistema de archivos mientras que una más era usada para realizar las operaciones de contabilidad y una última como coordinadora de las anteriores. Esto dio origen a la idea errónea de que en el cómputo distribuido sólo se realizan tareas que no están relacionadas y en el cómputo paralelo todas las tareas están estrechamente relacionadas con el fin de resolver un problema muy concreto.

Para finales de los años 80 y principios de los 90 las computadoras se volvieron más accesibles y muchas instituciones de carácter público y privado comenzaron a equiparse con equipos nuevos y a enlazarlos por medio de redes de área local con fines muy variados.

Con los avances de la tecnología de computadoras se inicio un fenómeno muy curioso que persiste en la actualidad, éste consiste que en algunos casos las capacidades de los distintos equipos son bastante sobradas para los requerimientos de los usuarios. Dicho de otra forma, el tiempo de uso del procesador es desperdiciado.

Viendo la oportunidad que representaba el aprovechamiento de estos recursos desperdiciados, se iniciaron una serie de investigaciones para resolver este problema.

En el año de 1993 surge el proyecto Condor [12] que si bien no fue el único ni el primero, ha sido uno de los más representativos. Creado en la Universidad de Wisconsin, el objetivo era formar un depósito de recursos computacionales aportados por cientos de estaciones de trabajo de universidades alrededor del mundo, con finalidades como simulaciones moleculares y diseño de motores a diesel entre otros.

Viendo las posibilidades, organizaciones de carácter distinto al educativo han aprovechado estos estudios. Advanced Micro Devices (AMD) utilizó técnicas similares usando cientos de computadoras durante las fases pico del diseño de sus microprocesadores K6 y K7. Estas computadoras se encontraban en los escritorios de los ingenieros de AMD y su uso principal era la verificación de los chips sólo cuando los ingenieros no las usaban [15]. Posteriormente

su principal competidor, Intel, inició el uso de estas técnicas en su línea de microprocesadores.

Hasta el año de 1996, sólo proyectos de participación privada o restringida se habían hecho presentes. La creación del proyecto GIMPS cambió este panorama al permitir una participación abierta a cualquiera que tuviera una computadora personal y acceso a Internet. Posteriormente en el año de 1997 se le uniría distributed.net, PiHex y otros más.

El año de 1999 marca un hito en proyectos de este estilo con la creación de SETI@home. Hasta antes de ese año los proyectos de cómputo distribuido basados en Internet tenían finalidades de carácter matemático, lo que inhibía la participación del público. La idea de cooperar en la búsqueda de inteligencia extraterrestre ha redituado en más de dos millones de participantes y desafortunadamente hasta el momento, ninguna evidencia de inteligencia extraterrestre. De cualquier forma este proyecto propició la creación de otros proyectos que si bien no han tenido una respuesta tan grande, hay que destacarles sus objetivos de entre los cuales se pueden mencionar la búsqueda de una cura para el cáncer, el estudio del genoma humano y la predicción del clima global.

1.3 ASPECTOS DE DISEÑO

El cómputo distribuido contempla tanto aspectos de software como de hardware. Básicamente los aspectos de hardware tratan de la organización de las computadoras involucradas y en particular, de la forma en que se conectan y comunican entre sí. Los aspectos de software contemplan la forma de aprovechar esta infraestructura de hardware disponible con un fin común.

Tanto el hardware como el software son factores que intervienen a la hora de diseñar e implementar un proyecto de cómputo distribuido. No es lo mismo diseñar e implementar un proyecto en una LAN de tipo Gigabit Ethernet formada por 100 equipos idénticos con procesadores a 3 GHz y 1 GB de RAM, que en una WAN de 10,000 equipos con velocidades y

capacidades que varían desde lo máximo a lo mínimo, la cual puede ser una red de computadoras conectadas por medio de Internet.

Debido a la gran variedad de proyectos que se pueden crear, existe también un número considerable de factores que hay que tomar en cuenta en el momento del diseño.
Para fines prácticos sólo se detallarán aquellos que intervengan en cierta clase de proyectos. Esta clase de proyectos se caracteriza por acelerar la solución de un problema dividiéndolo en procesos independientes ejecutándose uno por computadora (un proceso es una pieza de código secuencial que se ejecuta en una unidad de procesamiento).
Este criterio es conveniente ya que las herramientas desarrolladas en esta tesis están optimizadas para este tipo de proyectos.

1.3.1 Modelos de programación

Los modelos de intercomunicación entre procesos también se conocen como modelos de programación [20] y tienen sus orígenes en las arquitecturas paralelas. Básicamente existen dos modelos de programación: memoria compartida y paso de mensajes.

La memoria compartida es un solo espacio de direcciones en el cual los procesos pueden escribir y leer información para uso propio o para los demás procesos. Ésta suele dividirse en memoria compartida total y memoria compartida parcial. En la primera todo el espacio de direcciones se encuentra disponible para cualquier proceso, mientras que en la segunda existen regiones privadas a las que sólo pueden acceder un número limitado de procesos.

El paso de mensajes es una forma de comunicación en la cual un proceso emisor envía una secuencia de bytes con significado al proceso receptor con el que desea comunicarse. Para que el método sea eficiente ambos procesos deben contar con estructuras que les permitan interpretar los mensajes de forma correcta.

A pesar de que el manejo de los mensajes por parte de los procesos representa una carga adicional, existen pocas restricciones en cuanto al tamaño de la información que pueden contener. Así, el paso de mensajes se convierte en una forma efectiva de transmitir grandes bloques de datos de un proceso a otro. Esto, aunado al gran costo de mantener una memoria

compartida en una arquitectura distribuida, hace que el modelo de programación más usado sea el paso de mensajes para el cómputo distribuido.

1.3.2 Planificación

La planificación consiste en decidir el orden en cual los cálculos de un programa serán ejecutados, y por cual proceso [17]. La planificación está relacionada con el balanceo de la carga. El balanceo de lo carga es el grado como el trabajo se distribuye entre los procesadores disponibles. Un programa se ejecuta más rápido cuando la carga está balanceada perfectamente, que es cuando cada procesador tiene una parte del trabajo total y se realiza el trabajo de forma que todos los procesadores completan sus labores asignadas al mismo tiempo. Una medida del desbalanceo de la carga es la razón de la diferencia entre los tiempos de término del primero y el último procesador.
Generalmente una arquitectura distribuida tiene más tareas que unidades de procesamiento (computadoras). Debido a esta condición es necesario implementar la planificación propia que es la planificación más eficiente. La planificación propia consiste en mantener un depósito central de tareas y que cada procesador obtenga una nueva tarea de ahí sólo cuando haya terminado de realizar una anterior.
La planificación propia también recibe el nombre de modelo maestro-trabajador, el cual se detallará a continuación

Modelo Maestro-Trabajador

En este modelo, un proceso denominado maestro, se encarga de asignar tareas a otros procesos, denominados trabajadores. Cuando un proceso trabajador completa una tarea, se envían junto con los resultados obtenidos, una solicitud de más trabajo al proceso maestro, el cual responde con uno de dos mensajes: un mensaje conteniendo la siguiente tarea por ser realizada o una señal indicando que todo el trabajo ha sido concluido [4][3][22].
En este modelo, los trabajadores no interactúan con otros, solamente con el maestro. El maestro se encarga de coordinar los datos de entrada y de salida.

El modelo lo podemos resumir en los siguientes puntos:

- Las tareas y los trabajadores son independientes uno de otro.

- El maestro divide el trabajo en tareas discretas y las coloca en un espacio privado.
- Los trabajadores obtienen tareas y entregan resultados por medio del maestro.
- El maestro almacena los resultados en su espacio privado.
- Las tareas pueden ser distribuidas sin ningún orden o de forma ordenada de acuerdo a criterios específicos (velocidad del trabajador, espacio, prioridad, etc.)

Las principales ventajas de este modelo son las siguientes:

- Los nodos no necesitan estar conectados permanentemente
- Si la tarea asignada a un nodo no se realiza, se puede reasignar fácilmente a otro trabajador.
- Facilidad de integración de un trabajador nuevo
- Código sencillo de desarrollar y fácil depurar.
- Escalabilidad

Este último punto es muy importante. Por ejemplo, se puede crear una estructura jerárquica donde existan un maestro primario y varios maestros secundarios. Los maestros secundarios se encargan de atender solicitudes de trabajadores simples. La función del maestro primario es atender solamente a los maestros secundarios, de hecho, el maestro primario ve a los maestros secundarios como sus trabajadores y es de esta forma como se facilita la administración de un gran número de trabajadores.

Existen claras desventajas respecto a este modelo, como pueden ser la falta de comunicación directa entre nodos, o el requerimiento de que la dependencia de datos sea mínima o inexistente, lo cual limita el tipo de proyectos de cómputo distribuido en el que puede ser usado.

1.3.4 Seguridad

Debido a que las redes de computadoras no son seguras por si mismas, es necesario emplear formas de proteger la información que circula en la red y al mismo tiempo hacerla privada, es decir, que sólo los involucrados puedan manipularla de forma adecuada. Esto es particularmente cierto para una red que utiliza como infraestructura Internet.

Los siguientes son unos ejemplos de los posibles riesgos que se pueden presentar:

- Robo de información (tareas y resultados) de los servidores o clientes.
- Robo de información de los datos en el medio de comunicación.
- Irrupción en los servidores para que éstos distribuyan código malicioso a los clientes.
- Ataques de negación de servicio.

Debido a que cada día se descubren vulnerabilidades nuevas, no existe un consenso de como debe implementarse la seguridad en una arquitectura distribuida, sin embargo, el cifrado de los datos en curso es una de las aproximaciones más usadas actualmente, ya sea de clave pública o simétrica.

El problema de la ejecución de código malicioso en los clientes se puede corregir por el uso de "una caja de arena", es decir, un ambiente de ejecución aislado del sistema. Sin embargo, estos ambientes no están muy desarrollados en la actualidad o su desempeño general deja mucho que desear.

El aseguramiento de los equipos servidores también es un elemento clave. Restringiendo su acceso y configurándolo de forma adecuada se pueden corregir la mayor parte de los problemas de seguridad en un proyecto de cómputo distribuido.

1.3.5 Tolerancia a fallos

Se dice que un sistema falla cuando no cumple con su especificación. Los sistemas de cómputo pueden fallar debido a problemas en algún componente, como puede ser un procesador, memoria, dispositivos de E/S, etc. En una arquitectura distribuida esto se hace patente cuando una computadora que la conforma deja de funcionar, es decir, no genera o transmite más datos, ya sea de forma temporal o permanente. Un buen método de planificación permite solventar estos problemas reasignando labores a otros equipos sin problemas. Este tipo de falla se conoce como *silente*.

Existe otro tipo de fallas, llamadas *bizantinas*, en las cuales, las unidades de procesamiento (computadoras) producen resultados erróneos, ya sea de forma accidental o intencional.

Los resultados erróneos accidentales se pueden deber a problemas de hardware y estos se resuelven por medio del uso de algoritmos de paridad o checksum. De esta forma se sabe cuando un resultado es erróneo y simplemente se descarta y se toman las medidas convenientes para corregir el problema.

Una falla bizantina intencional requiere de un enfoque diferente de solución. Su origen generalmente se debe a actos mal intencionados de personas con conocimientos suficientes para burlar mecanismos como la paridad[4] o el checksum[5]. Existen formas de corregir este tipo de fallas como pueden ser la autenticación, cifrado de datos, ofuscación o técnicas de redundancia y aleatoriedad.

Las técnicas de redundancia permiten revisar los resultados que cada trabajador entrega y compararlos con otros que realizaron la misma labor. Si existe un número suficiente de trabajadores buenos, se puede entonces identificar a los que producen errores de forma intencional.

La aleatoriedad se usa para seleccionar un trabajador al azar de todos los disponibles y asignarle una labor de la cual se conoce de antemano el resultado. De esta forma se fuerza a los trabajadores a producir resultados correctos la mayor parte del tiempo.

1.4 EJEMPLOS DE PROYECTOS DE CÓMPUTO DISTRIBUIDO

Los proyectos de cómputo distribuido han tenido como representante más conocido el proyecto de la Instituto SETI de la Universidad de California Berkeley denominado SETI@home. Este proyecto se inició en 1995 y estuvo finalmente a disposición del público en 1999 con la finalidad de descubrir patrones inteligentes en los datos provenientes de un radiotelescopio y en la actualidad tiene registrados a más de 3 millones de "donadores de tiempo computacional", que en conjunto suman una capacidad computacional de más de 15

---

[4] Un algoritmo de paridad se encarga de detectar errores de transmisión de un byte de datos por medio de un bit adicional.

[5] El checksum es una suma de verificación de errores. Se utiliza para controlar todo un conjunto de datos transmitidos o almacenados y no solamente un byte.

TFLOPS[6](En el año 2001, ASCI Q era la supercomputadora más poderosa y tenía una capacidad de 10 TFLOPS)

El éxito de este proyecto creo un interés generalizado en esta clase de cómputo, estimulando la creación de proyectos con objetivos distintos, pero en esencia, la forma de operación es la siguiente:

1. Un individuo descarga de alguna página de Internet un software de aplicación, denominado a veces cliente, que generalmente es un protector de pantalla y lo instala en su computadora. Este software puede ejecutarse como un proceso de baja prioridad de forma permanente o de prioridad normal cuando el usuario no este usando ni el teclado ni el ratón, a modo de protector de pantalla.
2. Cuando el software cliente se ejecuta revisa si tiene operaciones o tareas por realizar, que en caso afirmativo, las realiza hasta completarlas, almacenando los resultados y mientras tanto, puede mostrar simples mensajes informativos o gráficos muy elaborados al usuario.
3. Si no hay más tareas por realizar el software cliente establece un canal de comunicación con una computadora servidora.
4. Si se cuenta con resultados de tareas previas, estos se envían al servidor.
5. Si no hay más resultados por enviar, se aprovecha el canal de comunicación para descargar más tareas y se cierra el canal de comunicación.
6. Se repite todo el procedimiento indicado desde el punto 2.

El esquema de la figura 1.1 ilustra mejor este proceso. La siguiente muestra es una parte de los proyectos de cómputo distribuido más importantes de la actualidad.

1.4.1 SETI@home

El programa de Búsqueda de Inteligencia Extraterrestre, también conocido como SETI, fue desarrollado en 1984 por el Instituto SETI de la Universidad de California Berkeley para dirigir investigaciones relacionadas con posibles formas de vida en el universo.

---

[6] Abreviatura de TeraFLOPS, lo que equivale a 1,099,511,627,776 operaciones de punto flotante

Figura 1.1 Esquema genérico de un proyecto de cómputo distribuido

Una comunidad de científicos considera que las ondas de radio pueden ser la forma más simple y eficiente de que formas de vida diferentes a las que se conocen realicen un contacto. Las razones son que algunas frecuencias de ondas de radio no requieren de mucho poder, y si la energía es concentrada en una frecuencia, puede ser fácilmente distinguida de otras. Sin embargo, no hay forma de saber si estas teorías son correctas. Esto aunado a que es necesario una gran capacidad computacional para procesar los datos, junto con un presupuesto muy limitado, motivó la creación del proyecto SETI@home.

En julio de 1996 los creadores de SETI@home, David Gedye y Craig Kasnoff, presentaron la idea de que los ciclos sin utilizar de las computadoras conectadas a Internet fueran aprovechados para el procesamiento de la información de los radiotelescopios. Esta idea fue ampliamente apoyada por lo que comenzó la creación del software cliente y servidor. El 17 de mayo de 1999, el sitio Web de SETI@home se abrió de forma oficial al público. La respuesta fue enorme, con incluso más de 100,000 usuarios interesados en el proyecto antes de que el sitio fuera abierto.

FUNCIONAMIENTO

El telescopio Arecibo en Puerto Rico obtiene todos los días datos consistentes en señales de radio. En un día pueden almacenarse hasta 35 Gigabytes de información en bruto. Debido a que es mucha información, en vez de enviarse por Internet, se utiliza correo regular para enviarlo a la UC Berkeley, donde se divide y se envía a todo aquel que tiene instalado el software cliente SETI@home. A través de Internet, las personas reciben trozos con "unidades de trabajo" de 0.25 megabytes.

Las computadoras "clientes" revisan las frecuencias de radio en el tiempo de inactividad del usuario, en especial las que tienen mucha energía dentro de un ancho de banda estrecho, lo cual podría indicar un mensaje extraterrestre.

Le toma a la mayoría de las computadoras entre 10 y 50 horas completar una unidad de trabajo. En ese tiempo, la computadora habrá realizado más de 175 mil millones de cálculos sobre una pieza de datos de 107 segundos. Entonces, la unidad de trabajo (ya procesada) se envía de regreso a los servidores de SETI y la computadora cliente recibe una nueva unidad de trabajo.

SETI@home tiene tres servidores, denominados Sagan, Asimov y Cyclops. Sagan envía las unidades de trabajo mientras que Cyclops y Asimov mantienen los resultados interesantes. Sagan almacena todas las unidades de trabajo recibidas de Arecibo, lo cual es útil si una unidad de trabajo específica se necesita enviar de nuevo. Esto pasa si una computadora cliente detiene su operación y nunca envía la unidad de trabajo completada.
Cabe recalcar que los datos enviados a los clientes son independientes uno de otro debido a que cada unidad de trabajo representa solamente algunas frecuencias de una pequeña sección del espacio, y para procesar los datos, las computadoras cliente no necesitan información de otras unidades de trabajo.

En la figura 1.2 se ilustra el funcionamiento de SETI@home. Por cierto, hasta la fecha, no se ha obtenido ninguna evidencia concluyente de haber obtenido un mensaje proveniente de seres extraterrestres.

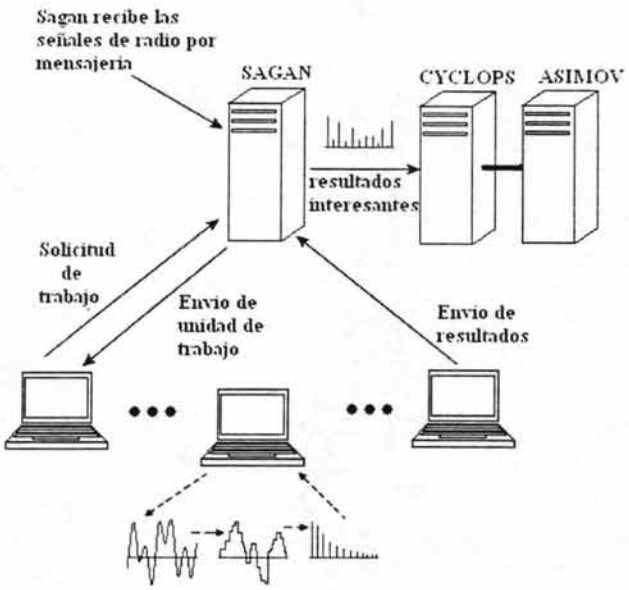

Figura 1.2 Esquema del funcionamiento de SETI@home

El software cliente de SETI@home se encuentra disponible para Windows (versiones 95/98/2000/NT/ME/XP), Macintosh y una versión especial sin interfaz gráfica (sólo línea de comandos) funciona en UNIX, Windows NT, OS/2, BeOS, Mac Os X Server, Open VMS. Una nueva versión mejorada de este software, denominada temporalmente SETI@home II, está siendo desarrollada con una herramienta de creación de proyectos de cómputo distribuido denominada BOINC. Esta última se analizará en un capitulo posterior.

1.4.2 Distributed.net

Las técnicas de criptografía tienen como finalidad evitar problemas de seguridad en sistemas computacionales. La criptografía se usa desde el envío de correo electrónico hasta transacciones monetarias, como es el caso del comercio electrónico. Como podemos notar, cada vez más actividades humanas dependen de los sistemas computacionales, por lo que la seguridad tiene un papel importante.

Una de las técnicas, el cifrado de datos de clave privada, consiste en modificar la información que se desea proteger de tal forma que sólo miembros autorizados puedan ver

su contenido. Esto involucra una serie de complejas operaciones matemáticas basadas en un clave de tamaño variable (el tamaño se mide en bits), de tal forma que solo los conocedores de esa clave tienen acceso a la información. Ya que el cifrado y descifrado de los datos requieren de muchos cálculos, el tamaño de la clave elegida es fundamental. Un tamaño exagerado redituaría en que solo supercomputadoras podrían manejar datos cifrados con esta clave, mientras que un tamaño mínimo haría que cualquier persona pudiera adivinar la clave, y por tanto, observar los datos.

RSA Security es una compañía responsable no solo de cifrar los datos de sitios Web comerciales y servidores, sino también del gobierno de los Estados Unidos de América. Esta compañía propone estándares en el tamaño de las claves de cifrado a utilizar, de tal forma que sean manejables para la tecnología actual.
Con el fin de asegurarse que sus estándares sean lo suficientemente seguros, esta compañía organiza concursos para medir la rapidez y la cantidad de recursos necesarios para adivinar la clave con la que se cifró un mensaje. En teoría, no debería ser posible para nadie adivinar la clave de un mensaje cifrado. El hecho de que este evento se presente indica que sus estándares no son lo suficientemente seguros y por lo tanto, cambiar a otros.

Uno sus concursos, denominado RC5-56 (el número 56 es debe a que el mensaje se cifró utilizando una clave de 56 bit, por lo que el número de claves posibles es 2^{56}) se lanzó en el año de 1997. Una organización denominada distributed.net aceptó el reto y 212 días después de iniciar sus actividades y con la colaboración de más de 4000 computadoras, encontraron la clave después de probar casi la mitad del espacio posible de combinaciones. Esta clave descifró el mensaje, el cual era "It's time to move to a longer keylength."(Es hora de moverse a una longitud de clave mayor).
Este hecho hizo que los estándares en longitud de claves abandonaran el tamaño de 56 bit por otros mayores. Un concurso similar posterior, denominado RC5-64 (claves de 64 bits) fue ganado nuevamente por distributed.net a mediados del 2002, después de 1757 días de iniciados los esfuerzos. Actualmente (hasta mayo del 2004) los esfuerzos de distributed.net están orientados al concurso RC5-72 (claves de 72 bits).

FUNCIONAMIENTO

El software de *distributed.net* se denomina Bovine. Cuando se descarga Bovine, el usuario puede elegir la clase de subproyecto en la que desea participar. Estos son actualmente OGR y RC5-72. A continuación se detallará el subproyecto RC5-72 a manera de ejemplo.
El objetivo del proyecto RC5-72 es encontrar la clave secreta de 72 bits usada para cifrar un mensaje. Este proyecto involucra una dificultad 256 veces superior al de encontrar un clave de 64-bit y 65,536 veces más difícil que una clave de 56 bit.
Le toma a una computadora entre 3 minutos y 12 horas procesar un bloque de claves. No es necesaria una conexión permanente a Internet, con sólo unos momentos de conexión cada pocos días, el software cliente puede contactar a los servidores proxy de *distributed.net* para obtener más bloques de claves. En el caso de no contar con una conexión a Internet al momento, el software Bovine puede tratar con bloques de claves creados al azar, de esta forma, no se desperdicia el tiempo.

La arquitectura creada por *distributed.net* para sus proyecto RC5-72 es de tipo piramidal y bastante eficiente. Lo más notable es que cada bloque de claves se envía sólo una vez a cada usuario único.
Existe un servidor de claves maestro, que se encuentra en la punta de la pirámide. Su función es llevar un registro de los bloques de claves o datos que necesitan ser procesados. De esta forma, el servidor de claves maestro distribuye los bloques de conforme se necesita y no vuelve a enviar los bloques que ya han sido revisados.

Debajo del servidor de claves maestro se encuentran los servidores de claves proxy. Estos actúan como mediadores entre el servidor maestro y los clientes. Estos servidores proxy solicitan grandes bloques de claves (superbloques) al servidor maestro. Estos superbloques son divididos y enviados a los clientes. Los clientes entonces pueden trabajar con los datos y regresarlos a los servidores proxy, los cuales a su vez los regresan al servidor maestro. En la figura 1.3 se ilustra el funcionamiento de distributed.net.

El subproyecto OGR es un más complicado que la búsqueda RC5-72. El objetivo es encontrar una regla óptima Golomb de 24 marcas, para lo cual hay que llevar a cabo un proceso sistemático complejo el cual no se detallará aquí. Por los demás aspectos, este subproyecto tiene un funcionamiento similar al RC5-72 en cuanto a tareas, servidores y clientes.

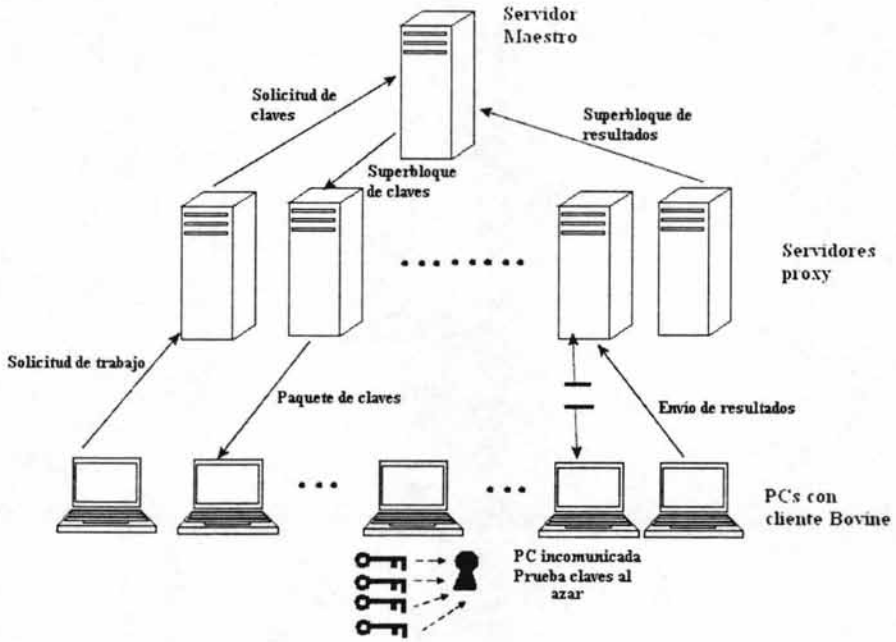

Figura 1.3 Esquema del funcionamiento de distributed.net

El código fuente de Bovine es parcialmente abierto. Ciertas secciones se mantienen secretas con el fin de evitar actos fraudulentos por parte de los usuarios. En su forma compilada y disponible para el público, Bovine es un software que tiene versiones que se ejecuta en los siguientes sistemas operativos: Windows (3.x/9x/2000/NT/ME/XP), PC-DOS, MS-DOS, Mac OS, Mac OS X, OS/2, GNU/Linux, BSD/OS, FreeBSD, NetBSD, OpenBSD, Solaris/SunOS, Amiga OS, SGI IRIX, Tru64 UNIX/DEC UNIX/OSF 1, BeOS, QNX, AIX, HPUX, NetWare Server, Acorn RISC OS.

1.4.3 GIMPS

GIMPS son las siglas en inglés de la "Búsqueda del Mayor número Primo Mersenne". Su objetivo, así como su nombre lo indica, es encontrar nuevos números primos Mersenne. Los número primos Mersenne pueden ser encontrados usando la ecuación 2^P-1. Cuando el

número resultante es primo, se dice entonces que P es un número primo Mersenne. Por ejemplo, $2^3-1 = 7$. Como 7 es un número primo, entonces 3 es un número primo Mersenne. Hasta el momento solo se han encontrado 38 primos Mersenne. La búsqueda de números primos Mersenne comenzó en 1996 con 40 personas y un poco más de 50 computadoras. Actualmente el proyecto cuenta con la colaboración de cientos de miles de personas y sus equipos alrededor del mundo, con lo cual ha sido posible descubrir nuevos números Mersenne cada año, que superan a los anteriores en cuanto a la cantidad de dígitos que tienen.

FUNCIONAMIENTO

El proyecto utiliza una prueba denominada Lucas-Lehmer, la cual es simplemente un prueba de la ecuación 2^P-1 en la que se varía el exponente P. Todas las computadoras participantes se dividen en tres categorías, de acuerdo a su velocidad.

En la primera categoría (las más veloces), el servidor del proyecto denominado Primenet asigna a las computadoras "la primera prueba de primalidad". Esto quiere decir que estas computadoras revisan rangos de números sin revisar usando la prueba Lucas-Lehmer y por lo tanto tienen la mayor posibilidad encontrar un nuevo número Mersenne.

Las computadoras de la segunda categoría (las de velocidad intermedia) tienen asignada la labor de revisar el trabajo realizado en las pruebas de primalidad. En esta categoría las posibilidades de encontrar un número primo Mersenne son muy bajas. La única posibilidad consiste en que en la primera prueba se haya pasado por alto un número primo Mersenne.

En la última categoría (las más lentas) no existe posibilidad de encontrar un número primo Mersenne. La labor consiste en realizar trabajo de factorización de apoyo para las pruebas de primalidad.

El proyecto GIMPS tiene la peculiaridad de que los resultados pueden ser enviados de forma manual por medio del correo electrónico, ya que así funcionaba en sus orígenes y se ha mantenido esta característica.

El código fuente de GIMPS esta disponible para su estudio y como invitación al público para que ofrezca mejoras, siguiendo las regulaciones de GIMPS, las cuales pueden consultarse en su página Web. En la figura 1.4 se ilustra el funcionamiento de GIMPS.

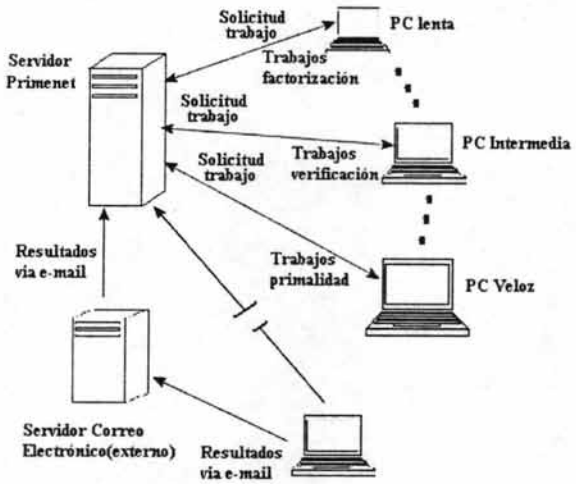

Figura 1.4 Esquema del funcionamiento de GIMPS

El software cliente de GIMPS está disponible para los siguientes sistemas operativos: Windows (versiones 3.1/9x/2000/NT/ME/XP), Linux, FreeBSD, OS/2, Mac OS X.

1.4.4 Otros proyectos

Ya se ha descrito el funcionamiento de tres de los proyectos distribuidos más grandes del mundo, en cuanto a número de usuarios. Sin embargo, no se puede dejar pasar por alto a otros proyectos [18]. Se mencionará de forma breve a un pequeño grupo de ellos y con esto comprobar, que el cómputo distribuido no se limita solamente a operaciones matemáticas.

FOLDING@home

Folding@home [14] es un proyecto que usa un protector de pantalla o una aplicación cliente basado en Mithral CS-SDK[23], con el fin de simular doblamiento de proteínas en un esfuerzo para entender mejor como las proteínas se ensamblan o doblan por si mismas. Intel apoya este proyecto a través de su programa Philanthropic Peer-to-Peer. Entre sus estudios

actuales se encuentra la enfermedad de Alzheimer, Hungtington y otros más. El software cliente está disponible para Windows, Linux y Mac OS X.

FightAIDS@Home

FightAIDS@Home[13] es un proyecto de investigación patrocinado por el Instituto de Investigación Scripps [29] con el fin de diseñar nuevos fármacos para combatir el SIDA. El software cliente creado por el proyecto está diseñado para predecir como pequeñas moléculas (sustratos o fármacos candidatos) se ligan a un receptor de una estructura tridimensional conocida. Este software se encuentra disponible para plataforma Windows, aunque se tienen planeadas versiones para Mac OS X y GNU/Linux.

MoneyBee

MoneyBee[25] es un protector de pantalla que utiliza redes neuronales para analizar los mercados de valores para predecir las tendencias futuras en los índices. El software cliente proporcionado se encarga entrenar su red neuronal con datos recientes del mercado de valores en un intento de realizar predicciones útiles. La participación en el proyecto es libre y todos los colaboradores tienen acceso a las predicciones en forma de gráficas y análisis técnicos. El software cliente se encuentra disponible para Windows y GNU/Linux.

climateprediction.net

Este proyecto [6] utiliza simulaciones Monte Carlo de gran escala para predecir el clima de la tierra de aquí a 50 años. Los primeros modelos de clima para una simulación de 45 años fueron completados en noviembre del 2003. El cliente del proyecto requiere de equipos poderosos. En particular, una unidad de trabajo toma hasta 6 semanas en completarse usando un CPU a 1.4 GHz. El cliente se ejecuta como una aplicación gráfica o como un proceso de fondo y está disponible solamente por el momento para plataforma Windows.

1.5 CONCLUSIONES

En este capítulo se presentó una breve introducción al complejo tema del cómputo distribuido, empezando por la definición que se utilizará a lo largo de este documento, pasando por su historia y categorización, así como la descripción general del funcionamiento de algunos de los proyectos de cómputo distribuido más relevantes. En particular, este último punto ilustra parte de la orientación que tendrá la infraestructura Éxigus.

Hoy en día existen varias infraestructuras para la implementación de proyectos de cómputo distribuido, tal como pretende Éxigus. En el capítulo siguiente capítulo, se estudian varias de ellas, con el objetivo de partir de una base sólida ya probada y poder experimentar, con nuevas características que sean de mayor beneficio a algún sector de usuarios en particular.

2. HERRAMIENTAS PARA CREAR PROYECTOS DE CÓMPUTO DISTRIBUIDO

Actualmente existen dos formas de crear un proyecto de cómputo distribuido: desarrollar todo el proyecto desde cero o utilizar algunas herramientas de desarrollo existentes.

Desarrollar el proyecto desde cero involucra la creación de las rutinas de programación básicas que se usarán en todo el proyecto. Entre las más importantes se encuentran:

- Administración de archivos y directorios: creación, borrado, renombramiento.
- Intercambio de información entre procesos.
- Control en la ejecución de procesos externos: iniciación, detención momentánea y terminación definitiva.
- Identificación del hardware del equipo anfitrión.
- Detección de señalizaciones del sistema operativo: inactividad del usuario, apagado del equipo.
- Compresión y descompresión de datos.
- Cifrado y descifrado de información, ya sea del tipo de clave pública o clave privada.
- Envío y recepción de información usando una red de comunicaciones de tipo TCP/IP y protocolos como HTTP y FTP.

Esto tiene varias desventajas, como puede ser un tiempo de desarrollo y prueba mayor al planeado, aunado a gastos mayores. Sin embargo, el desempeño, seguridad, tamaño y simplicidad del producto final compensan gran parte de las desventajas.

Debido a que no todos tienen los recursos adecuados (tiempo, conocimientos y dinero) para crear un proyecto de cómputo distribuido desde sus raíces se crearon herramientas especializadas para subsanar estas carencias de recursos en algún determinado grado. Estas herramientas reciben nombres como COSM, BAYAHNINHAN, FIDA y BOINC entre otros y se pueden dividir en dos tipos. En el primer tipo se encuentran las que consisten en un conjunto de bibliotecas de funciones especializadas sobre las cuales se diseña e implementa la solución a un problema particular. El segundo tipo de herramientas se caracterizan por ser una especie de plantillas, es decir, un molde en el que sólo basta codificar el algoritmo de solución adecuado; de la distribución, transporte, manejo de protocolos, seguridad y tolerancia a fallos se encarga la herramienta.

Ambos tipos de herramientas tienen ventajas y desventajas. En el caso de las herramientas tipo biblioteca se requieren de amplios conocimientos para resolver aspectos como seguridad y tolerancia a fallos, por ejemplo, pero el producto final queda hecho a los intereses de los desarrolladores.
Las principal ventaja de una herramienta tipo plantilla[7] es el ahorro de código, además de que no son necesarios conocimientos amplios del tema. Sin embargo, algunas necesidades de los desarrolladores no se ven cubiertas por este tipo de herramienta.

En vista de esto, muchas herramientas actuales están tomando un enfoque de tipo híbrido, es decir, se combina la sencillez de una solución tipo plantilla, pero aparte se proporcionan los servicios necesarios para aquellos desarrolladores que deseen mejorar sus productos.

A continuación se verá un análisis de las principales herramientas disponibles en la actualidad para facilitar la creación de un proyecto de cómputo distribuido.

2.1 ACERCA DE LOS ANÁLISIS REALIZADOS

Básicamente las herramientas aquí presentadas ofrecen soluciones de tipo maestro-trabajador, la mayoría de ellas optimizadas para el ambiente que se presenta en una red

[7] En inglés el término es framework, del cual no hay una traducción muy apropiada en español

como es Internet. Se procuró seguir el mismo esquema de análisis para las herramientas aquí analizadas, sin embargo, puede darse el caso de que la información requerida no esté disponible. Este análisis se divide en los siguientes seis aspectos principales:

Información técnica. La parte de información técnica tiene la finalidad de indicar aspectos sencillos relativos a la herramienta, los cuales son los siguientes:

1. Sistema o sistemas operativos de ejecución de la aplicación maestro: se enumeran los sistemas operativos que dan soporte a las aplicaciones tipo maestro
2. Sistema o sistemas operativos de ejecución de la aplicación trabajador: se señalan los sistemas operativos en los que se ejecutan las diversas aplicaciones clientes.
3. Lenguaje o lenguajes de programación: se listan los lenguajes de programación en los que es posible crear el proyecto.
4. Bibliotecas: se especifican aquellas sobre las cuales está construida la herramienta
5. Licencia: se indica la licencia bajo la cual se distribuye la herramienta, lo cual determina las condiciones de uso y costo.
6. Software adicional: se muestran los paquetes de software adicional ajenos a la distribución de la herramienta que son necesarios u opcionales para su correcto o mejor funcionamiento.

Arquitectura. Un proyecto de cómputo distribuido está compuesto por varias clases de software que se ejecutan en distintos equipos. Aunque básicamente existen dos clases (la clase maestro y la trabajador) puede darse el caso de que la herramienta maneje clases adicionales o aun más, que las clases tengan funciones omitidas o agregadas.
La finalidad de esta parte del análisis es mostrar los distintos roles que pueden tomar las aplicaciones involucradas, así como mostrar la forma en que interactúan entre sí y los datos que dan soporte a cada una de ellas.

Características especiales. Este aspecto del análisis puntualiza los aspectos más relevantes de la herramienta en cuestión, además que se incluye información que no corresponde a los otros aspectos del análisis.

Seguridad y tolerancia a fallas. Ningún software de aplicación está exento de riesgos y vulnerabilidades, a menos de que se tomen las medidas adecuadas. Esta parte cubre las

medidas adoptadas por la herramienta para proteger los proyectos creados. De forma adicional se indican los mecanismos de tolerancia a fallos implementados por omisión.

Creación de una aplicación. En este punto del análisis se muestra de forma breve los pasos a seguir para crear un proyecto de cómputo distribuido utilizando la herramienta en cuestión y de esta manera formarse una idea de lo simple o complicado que puede resultar el uso de esta herramienta.

Evaluación. Finalmente en evaluación se muestran los aspectos más relevantes de la herramienta en cuestión, los cuales sirven como distintivo en comparación con las otras. También se indican las ventajas y desventajas tanto del uso de la herramienta como del tipo de proyecto creado.

2.2 FIDA

FIDA [11] es una herramienta para la creación de aplicaciones distribuidas independientes y con un diseño que simplifica la construcción de aplicaciones distribuidas entre múltiples plataformas. El propósito de FIDA es el aprovechamiento del tiempo perdido de CPUs en PCs conectadas a través de Internet. Esta herramienta fue creada en la Universidad de Washington con apoyos de la NSF (Fundación Nacional de Ciencia) de los E.U.A.

2.2.1 Información técnica

Los detalles técnicos más relevantes se resumen a continuación:

Sistemas operativos para la ejecución de la aplicación maestro:

Windows 9x/Me/NT/2000/XP, GNU/Linux, MAC OS X

Sistemas operativos para ejecución la aplicación trabajador:

Windows 9x/Me/NT/2000/XP, GNU/Linux, MAC OS X

Lenguajes de programación que soporta: cualquiera que admita llamadas desde un objeto creado en lenguaje C

Bibliotecas usadas: Mithral client/server, Cryptlib
Licencias: Phase 1 COSM software License, Cryptlib
Software adicional: ninguno

2.2.2 Arquitectura

En FIDA se destacan dos aplicaciones o roles: cliente y servidor. Los clientes se encargan de solicitar tareas y enviar los resultados, mientras que el servidor tiene la función de coordinar la asignación de trabajo a los clientes solicitantes y recolectar los resultados de los mismos. No existe comunicación entre clientes y estos deben ser responsables de iniciar y terminar la comunicación con el servidor. Debido a estas características, los participantes forman una topología estrella con el servidor en el centro y el modelo de programación es el paso de mensajes.

La información que circula entre un cliente y servidor es de dos tipos: tareas y módulos de aplicación cliente. Los módulos de aplicación cliente son bibliotecas de carga dinámica (lo que en Win32 son los archivos *.dll y en UNIX los lib*.so, donde el símbolo '*' representa cualquier denominación) que implementan los algoritmos de solución de las labores encomendadas. Por último, las tareas son secuencias de bytes sin requerimientos de formato que se pasan como parámetros a los módulos de aplicación. En la figura 2.1 se muestra el diagrama de la arquitectura FIDA.

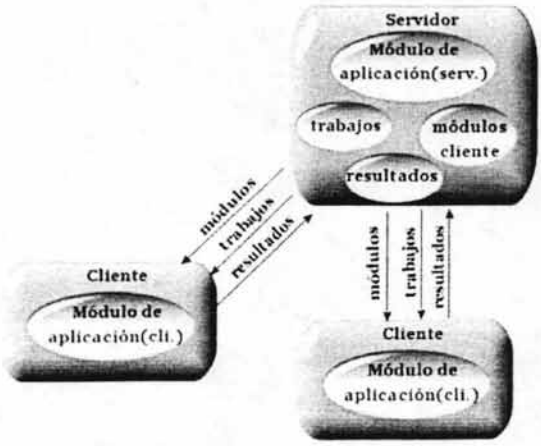

figura 2.1 Arquitectura FIDA

El protocolo de comunicación entre un cliente y el servidor es el siguiente:

1.- Un cliente abre una conexión de red con el servidor y envía un mensaje denominado REQUEST para solicitar una asignación de trabajo. El mensaje REQUEST incluye información acerca del cliente como puede ser el tipo de sistema operativo, el modelo de procesador, la versión de FIDA, el nombre del módulo de aplicación actual, si es que hay alguno.

2.- El servidor atiende la solicitud realizando una asignación de trabajo enviándolo como parte de un mensaje denominado ASSIGN al cliente, sólo si el modulo de aplicación del cliente esta actualizado. En caso contrario, el servidor envía al cliente mensaje UPDATE. Este mensaje contiene el módulo de aplicación apropiado.

3.- Si el servidor le responde al cliente con un mensaje ASSIGN, el cliente cierra la conexión con el servidor y comienza su labor de resolver las tareas asignadas.

4.-Cuando el cliente termina sus labores, abre una nueva conexión con el servidor para informar de los resultados.

5.- Si el servidor le responde al cliente con un mensaje UPDATE, el cliente detiene cualquier modulo de aplicación cargado, guarda el modulo actualizado y reinicia el nuevo módulo de aplicación.

6.- El cliente entonces envía un mensaje REQUEST y el proceso se repite desde el punto número 1

2.2.3 Características especiales

FIDA cuenta con las siguientes características especiales:

- Los clientes realizan sus labores por medio de la llamada de funciones predeterminadas en bibliotecas de carga dinámica
- Las bibliotecas pueden ser actualizadas durante el transcurso de vida de un proyecto
- Se usan firmas digitales para garantizar la integridad de los módulos ejecutables y la autenticidad de origen
- Los clientes pueden trabajar como protectores de pantalla o simples aplicaciones
- El servidor se encarga de generar las tareas al momento de una solicitud

- No requiere de software especializado adicional

2.2.4 Seguridad y tolerancia a fallos

El mecanismo de seguridad empleado por FIDA se basa en cifrado de datos de clave pública y su funcionamiento es el siguiente:

1.- El servidor genera un valor hash de 160 bits basado en el archivo de aplicación que será enviado. La función usada es SHA1.
2.-El servidor entonces cifra este valor hash usando la clave privada proveniente de un sistema criptográfico de clave pública (RSA en particular). Este cifrado se convierte en la firma del archivo.
3.-Cuando el servidor FIDA transfiere el archivo de aplicación, también se transfiere el contenido de la firma, todo en el mismo mensaje.
4.-El cliente, usando la clave pública del servidor, descifra la firma digital.
5.-El cliente entonces ejecuta la misma función hash sobre el archivo recibido, omitiendo la firma.
6.-El cliente compara este valor hash con el de la firma digital descifrada y los compara. Si ambos son iguales, la firma es válida y el archivo de aplicación es aceptado, en caso contrario, se desecha.
El protocolo descrito anteriormente protege los sistemas clientes permitiendo revisar la validez del código que reciben. Para hacer uso de este proceso de seguridad, los módulos clientes deben ser instalados junto un archivo (certFile) que contenga el certificado de la clave pública del servidor.
Con la finalidad de proporcionar esta seguridad adicional, FIDA utiliza una biblioteca de criptografía de carácter público conocida como Cryptlib. Por cuestiones legales (restricciones de importación y exportación), esta biblioteca no puede ser incluida en la distribución de FIDA.
Solo falta señalar que ningún otro tipo de datos que son enviados por la red tienen este esquema de seguridad, por lo que, de ser necesario, debe implementarse la seguridad de forma manual.

Este mecanismo tiene las siguientes propiedades:

1) Los clientes no ejecutarán código que no venga del servidor

2) Los binarios ejecutables en transito quedan protegidos contra alteración

La principal desventaja de este mecanismo de seguridad consiste en que los datos que circulan del cliente al servidor no están protegidos, por lo que pueden ser alterados o robados. En la figura 2.2 se muestra el diagrama a bloques de este mecanismo.

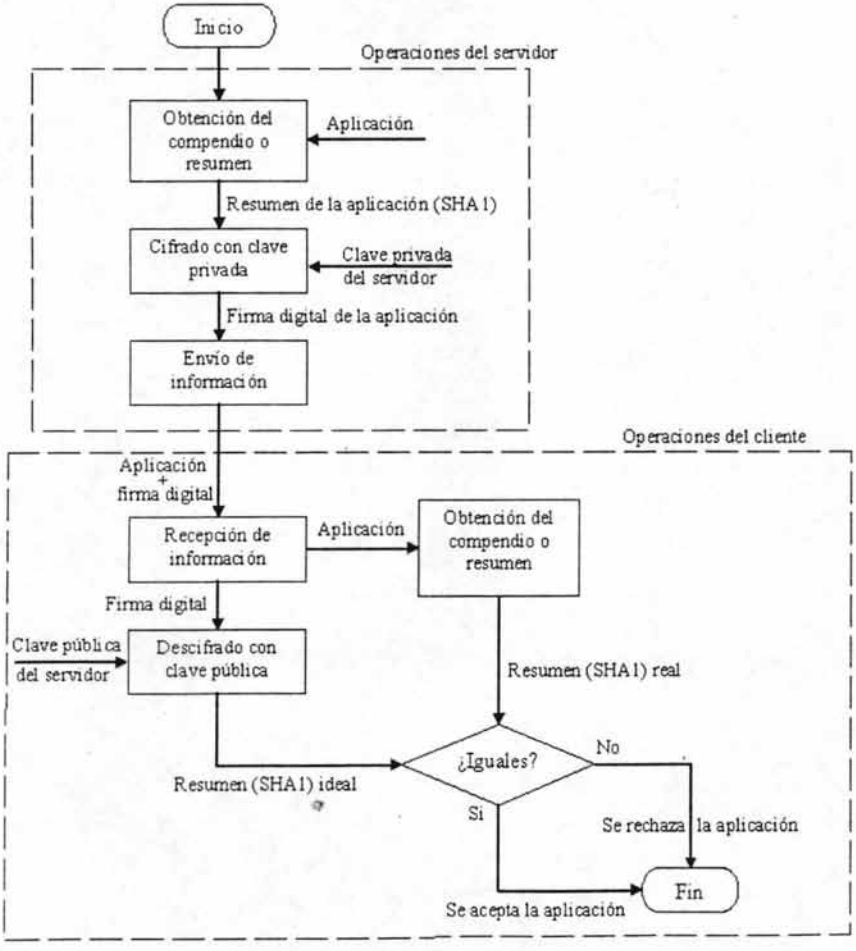

Figura 2.2 Diagrama de bloques del proceso de firma digital

Para terminar este punto del análisis, cabe señalar que esta herramienta no cuenta con mecanismos de tolerancia a fallas intencionales, por lo que se deben implementar de forma manual en caso de requerirse.

2.2.5 Creación de una aplicación

1.- Se descarga el código fuente FIDA y se compila. Como la mayoría de las herramientas que ponen a disposición el código fuente, incluyen archivos para automatizar el proceso de compilación (archivos make). Es necesario incluir la biblioteca Mithral client/server y de forma opcional la biblioteca cryptlib para satisfacer las dependencias necesarias.

2.- Se crean los módulos de carga dinámica que implementan la lógica necesaria para resolver las tareas encomendadas. Son cuatro funciones para el módulo servidor y cuatro para el módulo cliente que hay que definir y compilarlas de acuerdo a la plataforma objetivo (archivos *.dll para Win32 y lib*.so para variantes de UNIX).

3.-Se crean los archivos de configuración adecuados, tanto para el servidor como para el cliente. Estos son archivos de texto plano con una sintaxis predeterminada.

4.-Se empaquetan y distribuyen los archivos de acuerdo a la función que se desea desempeñar (servidor o cliente)

2.2.6 Evaluación

FIDA fue diseñado pensando en la sencillez de implementación y fácil actualización del algoritmo de solución, lo cual tiene sus ventajas y desventajas.
Las ventajas radican en que esta es una de las herramientas más simples de utilizar de las que se encuentran disponibles y en la que crear un proyecto completo toma poco tiempo. La actualización dinámica permite utilizar la base de computadoras clientes disponible para que trabajen en nuevos problemas. También se pueden crear versiones optimizadas del algoritmo de solución, dependiendo de las nuevas arquitecturas de procesadores

emergentes, por lo que la obsolescencia de una aplicación cliente es menor a la de una solución basada en compilación estática.

Las desventajas de esta herramienta son para tomar en cuenta. No se apoya en un administrador de base de datos para llevar control de la repartición de tareas y recepción de resultados. No hay seguridad en la información que circula del cliente al servidor y su carencia de un mecanismo de tolerancia a fallos puede representar un grave inconveniente en ambientes de escaso control como puede ser una red de un gran número de computadoras conectadas por Internet. Precisamente su falta de escalabilidad limita la participación que se pudiera obtener.

2.3 MODELS@home

MODELS@home [24] fue creado en el CMBI (Center for Molecular and Biomolecular Informatics, o Centro de Informática Molecular y Biomolecular) en Holanda tomando en cuenta dos hechos importantes de la modelación molecular:

1) El modelado frecuentemente requiere de una colección completa de programas de aplicación para obtener un resultado. Adaptar todos estos programas para una ejecución paralela es una labor prohibitiva en cuanto al tiempo consumido o simplemente imposible (en el caso de que el código fuente no se encuentre disponible)

2) Las aplicaciones de modelado raramente son de grano basto e insensibles a la pérdida de trabajos o resultados. En la mayoría de los casos, todos los trabajos deben ser completados antes de que un resultado esté disponible y muchas veces el verdadero factor limitante no es el número de computadoras sino el tiempo.

MODELS@home fue creado originalmente para subsanar los gastos que representaba la adquisición de una supercomputadora para fines de modelado molecular, sin embargo, los diseñadores de ésta pronto se dieron cuenta que podía utilizarse con finalidades diferentes por lo que decidieron ponerla a disposición del público.

2.3.1 Información técnica

Los detalles técnicos más relevantes se resumen a continuación:

Sistemas operativos para la ejecución de la aplicación maestro:

Windows 9x/Me/NT/2000/XP, GNU/Linux

Sistemas operativos para ejecución la aplicación trabajador:

Windows 9x/Me/NT/2000/XP, GNU/Linux

Lenguajes de programación que soporta: C, C++ y dependiendo del ambiente trabajador, Python

Biblioteca usada: SDL

Licencias: GPL, LGPL

Software adicional: ninguno

2.3.2 Arquitectura

En MODELS@home se destacan tres aplicaciones o roles: cliente trabajador, servidor y cliente supervisor. Cada uno de estas aplicaciones esta compuesta por módulos. Mientras que la aplicación cliente trabajador esta formada por tres módulos denominados Exec (módulo de ejecución de aplicación), ID (módulo de detección de inactividad), y ScrSaver (módulo gráfico de protector de pantalla), las aplicaciones servidor y cliente supervisor sólo contienen un módulo cada una, los cuales son Job Scheduler (planificador de trabajos) y Job Spawning (productor de trabajos) respectivamente. El diagrama de esta arquitectura se muestra en la figura 2.3

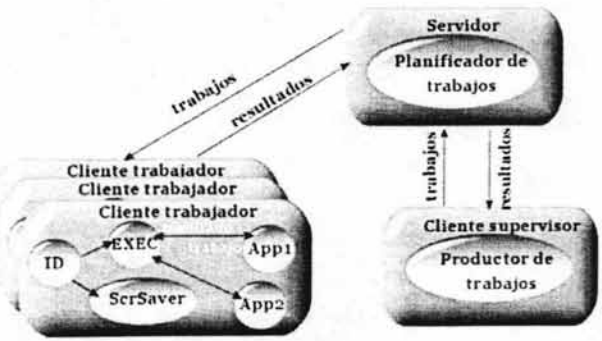

Figura 2.3 Arquitectura de MODELS@home

El cliente supervisor lleva un registro del trabajo que se debe realizar, mas no lo hace. Su labor consiste en cortar el trabajo en piezas y enviar los trabajos individuales al planificador de trabajos (Job Scheduler) por medio de una interfaz de funciones en lenguaje C o una clase Phyton. Esta labor se realiza bloqueando la cola central de trabajos (el cual es un archivo llamado "cluster.job") en el Servidor, agregando las nuevas labores, guardando el archivo y desbloqueándolo. Al final de este proceso, la cola central de trabajos contiene las indicaciones necesarias para determinar que labor debe ser realizada por un aplicación específica y en que orden.

Como sugiere el nombre, el cliente trabajador es el encargado de realizar las labores. Inicialmente el módulo de detección de inactividad (ID) se encuentra latente en el equipo donde se encuentra instalado. Tan pronto como se cumple un periodo predeterminado de inactividad se inicia el módulo gráfico de protector de pantalla (ScrSaver) y el módulo de ejecución (Exec). El módulo de ejecución crea un directorio vacío y contacta al servidor por medio del envío de un mensaje especial que contiene de forma adicional las fechas de los archivos que pudieran ser actualizables. El servidor revisa su cola de trabajos pendientes y en caso de encontrar alguno envía al trabajador cliente la descripción de las labores a realizar (la cual consiste en el nombre de la aplicación a ejecutar y los parámetros de línea de comandos), archivos de soporte y actualizaciones (en caso de que se encuentren disponibles). Una vez recibida esta información el módulo de ejecución inicia la aplicación respectiva y espera hasta que el trabajo sea concluido para enviar los resultados de regreso al servidor.

El servidor es el enlace entre los clientes trabajadores y el supervisor. Esta aplicación administra la cola de trabajos (que consiste en un archivo denominado "cluster.job") y distribuye las labores de acuerdo a su prioridad. También se encarga de manejar la transferencia de los archivos de trabajo desde el directorio del cliente supervisor al directorio del cliente trabajador (por medio TCP/IP) y viceversa en el caso de los resultados. De forma adicional, el servidor almacena las ultimas actualizaciones de archivos y las transmite a los clientes trabajadores para reemplazar versiones antiguas. Estos archivos pueden ser de cualquier tipo, incluso binarios ejecutables.

La última función del servidor consiste en revisar el estado actual de los clientes trabajadores. Si un cliente no envía un mensaje de estado dentro de un periodo de tiempo determinado, se asume una falla y el trabajo se retransmite a otro cliente.

2.3.3 Características especiales

MODELS@home cuenta con las siguientes características especiales:

- Los clientes realizan sus labores por medio de la ejecución controlada de binarios ejecutables por sí mismos
- El cliente sólo trabaja como protector de pantalla
- Las dos aplicaciones que forman el lado servidor pueden ejecutarse en equipos diferentes
- Se pueden actualizar los archivos que forman el cliente durante el ciclo de vida del proyecto
- Los clientes tienen un límite de tiempo para realizar un trabajo
- No requiere de software especializado adicional

2.3.4 Seguridad y tolerancia a fallos

MODELS@home no cuenta con mecanismos de seguridad especiales. De hecho, esta herramienta está orientada a ambientes de tipo LAN. Las razones de esto obedecen a que los problemas de modelación molecular (el motivo de origen de esta herramienta) no requieren de un número grande de computadoras trabajando en conjunto. Otra razón es que el software que se encarga de realizar el trabajo pudiera no ser libremente distribuido.

Al no utilizar siquiera un cifrado de datos, las ventajas que representa la carencia de un mecanismo de seguridad (como pueden ser una mayor facilidad de transmisión de datos y omisión de la sobrecarga en las aplicaciones debido a mecanismos de autenticación) se ven sobrepasadas por las riesgos de seguridad que involucraría utilizar esta herramienta para proyectos en ambientes de red como Internet.

Respecto a la tolerancia a fallos, esta herramienta ofrece mecanismos de redistribución de trabajo en caso de problemas en algún equipo de la red, mas no soporta fallas intencionales.

2.3.5 Creación de una aplicación

Nota.- Aunque esta herramienta soporta ambientes Windows y GNU/Linux, es necesario que la aplicación servidor y el cliente supervisor se ejecuten bajo el mismo sistema operativo (el cliente trabajador puede usarse de forma indistinta en cualquiera de estos sistemas operativos). Para fines prácticos, el proceso aquí descrito asume que el servidor y el cliente supervisor están instalados en el mismo equipo con sistema operativo GNU/Linux y el cliente trabajador funciona bajo sistema operativo Windows.

1.- Se descarga el archivo comprimido "cluster.zip" desde la página principal de distribución. Esta herramienta se distribuye de forma binaria por lo que no es necesario un proceso de compilación.

2.- El archivo "cluster.zip" se descomprime en el equipo que funcionará como servidor. A continuación se edita el archivo de configuración "cluster.cnf" de forma que la dirección IP del equipo servidor sea la correcta y por último se copian ciertas bibliotecas compartidas libSDL al directorio /usr/lib en caso de no tenerlas instaladas.

3.-A continuación se crea la aplicación cliente supervisor. Esta aplicación debe dividir las labores en forma de archivos, actualizar la cola de trabajos y almacenar los resultados que le envíe el servidor.

4.-Se crea la aplicación cliente trabajador, la cual implementa el algoritmo de solución en un archivo de lenguaje C o C++ y se compila en forma de binario ejecutable. Es importante que este aplicación maneje de forma adecuada los trabajos que se le indiquen desde la línea de comandos.

5.- Se empaca la aplicación cliente trabajador junto con algunos archivos de dependencias y un script de instalación para el correcto copiado de los archivos.

6.- Para iniciar el servidor, se escribe en la línea de comandos "cluster –ser –new" y se presiona la tecla "Enter". Una vez realizado lo anterior se inicia el programa cliente supervisor invocándolo desde la línea de comandos.

7.- Finalmente, cuando los clientes trabajadores detecten inactividad, se inicia el protector de pantalla y el proceso de resolución de tareas.

2.3.6 Evaluación

Una de las ideas más importantes del cómputo distribuido es hacer que aplicaciones que fueron concebidas para su ejecución en una sola computadora, mejoren su desempeño al "convertirlas" a aplicaciones distribuidas, esto sin modificar la propia aplicación. Más que convertirlas lo que en realidad se hace es crear un ambiente de ejecución controlado en el que se puedan aprovechar de forma conjunta los recursos de varios equipos.

MODELS@home utiliza este tipo de ideas en su diseño e implementación, lo cual representa una de las principales ventajas de esta herramienta. Teniendo el software de aplicación apropiado, sólo basta escribir la aplicación supervisora para poner en marcha un proyecto de cómputo distribuido. Esto nos da un indicio de la sencillez de uso de esta herramienta. Otro punto destacable es que no hay lineamientos especiales en caso de implementar el algoritmo de solución por cuenta propia, solo basta con manejar los parámetros de la línea de comandos de forma adecuada.

Esta herramienta tiene serios inconvenientes en caso de que se desee utilizarla en un ambiente de red tipo Internet. Al no contar con ningún mecanismo de seguridad se vuelve necesario implementar alguno para proteger la integridad del proyecto. Su carencia para solventar fallas intencionales también se vuelve parte de los agregados que debe realizar el desarrollador.
Como los mismos diseñadores de la herramienta lo comentan, esta herramienta es un serio candidato para la realización de proyectos de cómputo distribuido en ambientes controlados de tipo LAN.

2.4 Q²ADPZ

Q²ADPZ [27] (pronunciado cuodpisi en inglés) es una implementación modular de una sistema multiplataforma, multiusuario y de código abierto para solicitudes de cómputo distribuido en una red TCP/IP.

Desarrollado por el Departamento de Ciencia de la Computadora e Información en la Universidad de Ciencias y Tecnología de Noruega, actualmente se le utiliza de forma interna para labores de investigación en áreas como visualización científica de gran escala, computación evolutiva y simulación de modelos complejos de redes neuronales.

2.4.1 Información técnica

Los detalles técnicos más relevantes se resumen a continuación:
Sistemas operativos para la ejecución de la aplicación maestro:
Windows 9x/Me/2000/XP, GNU/Linux, FreeBSD, SunOS, IRIX64
Sistemas operativos para ejecución la aplicación trabajador: Windows 9x/Me/2000/XP, GNU/Linux, FreeBSD, SunOS, IRIX64
Lenguajes de programación que soporta: C,C++
Bibliotecas usadas: libcurl, libcrypto, bzip2
Licencia: GPL
Software adicional: servidor Web Apache

2.4.2 Arquitectura

En Q²ADPZ se destacan tres aplicaciones o roles: maestro, cliente y esclavo. En un sistema típico Q²ADPZ se encuentra un maestro, varios clientes y muchos esclavos (el número de esclavos sobrepasa al de los clientes). Estas aplicaciones se comunican entre sí por medio de mensajes en formato XML, los cuales se envían por medio de protocolo UDP. Archivos binarios como ejecutables se envían por medio de TCP a nivel de la capa de transporte y por

http a nivel de aplicación, esto de acuerdo al modelo OSI.[8] El diagrama de la arquitectura se muestra en la figura 2.4

La entidad cliente proporciona una forma de interacción entre los usuarios comunes y el sistema. Consiste en una biblioteca de servicios cliente y un módulo adicional definido por el usuario. Este módulo debe encargarse de comunicarse con la aplicación maestro, controlar e iniciar las tareas y obtener los resultados. La forma de implementar este módulo es: 1) utilizando llamadas a funciones de la biblioteca de servicios cliente (en este caso el módulo creado recibe el nombre de aplicación cliente usuario) o 2) utilizar un módulo ya implementado (denominado cliente universal) que se controla por medio de archivos de sentencias en formato XML.

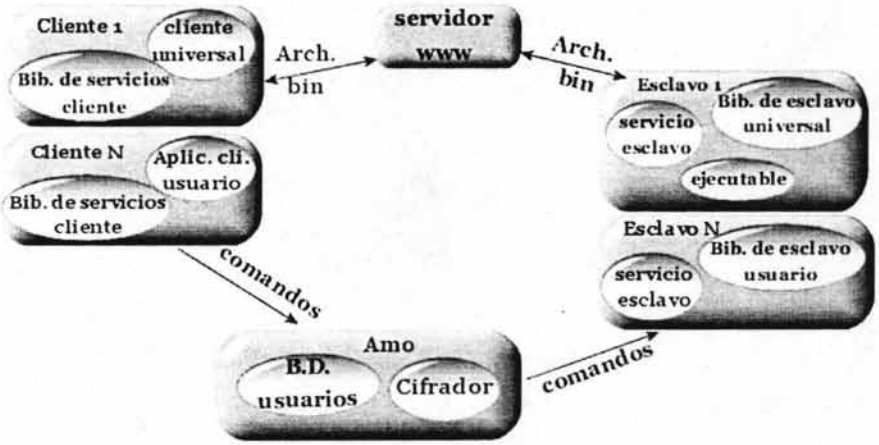

Figura 2.4 Arquitectura de Q²ADPZ

(nota.- Arch. bin. representa a archivos binarios como ejecutables, bibliotecas, archivos de entrada y de salida y los comandos son instrucciones especiales para controlar la asignación y ejecución de las tareas)

En cualquier caso, se debe o deben de crear los programas usuarios esclavos, los cuales consisten básicamente en la implementación de un algoritmo de solución en forma de bibliotecas de enlace dinámico, archivos ejecutables o programas interpretados. De forma adicional se deben preparar los archivos binarios que se utilizarán como entrada para los programas usuarios esclavos.

---

[8] El modelo OSI es una abstracción de la comunicación en redes dividida en 7 niveles.

Con todos estos datos se crean trabajos, los cuales son un grupo de tareas. Una tarea consta del programa de usuario, archivos de entrada y archivos de configuración. Los archivos de configuración (en formato XML) indican entre otras cosas el orden en que se deben desempeñar las tareas, cuales son los archivos de resultados, el tiempo máximo de espera y los requerimientos mínimos de hardware para que la tarea se lleve a cabo.

Una vez que los trabajos están definidos, se inicia la comunicación con la aplicación maestro, proceso en el cual se debe llevar a cabo una autenticación como usuario legitimo del sistema. Una vez que el canal de comunicación se encuentra listo, se envían los trabajos y la aplicación cliente entra en un estado de espera para recibir los resultados que le entregue la aplicación maestro.

La aplicación esclavo se encarga de realizar las tareas que le asigne el maestro. La aplicación está compuesta por el servicio esclavo y dependiendo de la tarea a realizar, una biblioteca proporcionada por el sistema (biblioteca de esclavo universal) o proporcionada por el usuario (llamada biblioteca de esclavo usuario).

El servicio esclavo tiene las siguientes funciones:

1) Envía de forma periódica mensajes indicando el estado de la aplicación esclavo (por ejemplo, resolviendo una tarea, ocioso, bloqueado, etc.) así como los recursos disponibles (memoria, cpu, software instalado, etc.)
2) Recibe las asignaciones de trabajo que le hace la aplicación maestro
3) Inicia la biblioteca de esclavo (usuario o universal) cuando hay labores pendientes o nuevas y el momento es conveniente (cuando el equipo no está en uso)
4) Le indica a la biblioteca de esclavo (usuario o universal) el momento en que debe terminar su ejecución y forzar esta condición en caso de falta de respuesta de la biblioteca.

La biblioteca de esclavo (usuario o universal) tiene las siguientes funciones:

1) Al momento de iniciarse revisa los archivos de configuración de tareas y descarga de un servidor Web de archivos aquellos que le hagan falta para realizar una tarea.
2) Una vez satisfechas las dependencias de archivos ejecuta en el orden adecuado el o los programas usuarios esclavos, utilizando de forma correcta los archivos de entrada indicados en la definición de la tarea.
3) Una vez que el código ejecutable ha sido concluido y no hay más labores pendientes, se envían los resultados a la aplicación maestro y la biblioteca se descarga de memoria.

Cabe destacar que la biblioteca de esclavo universal sólo se puede usar con programas ejecutables o interpretados. En caso de querer ejecutar una biblioteca de carga dinámica se debe usar la biblioteca de esclavo usuario

La última aplicación, el maestro, sirve como puente de unión entre las aplicaciones cliente y esclavo, debido a que éstas no pueden comunicarse de forma directa (salvo casos muy especiales, explícitamente creados para ese fin). De forma detallada, el maestro se encarga de monitorear a todos los esclavos, de esta forma, se tiene una visión general de los recursos disponibles en el sistema como conjunto. También se encarga de aceptar las solicitudes de tareas por parte de los clientes y asignarlas para su solución a los esclavos (dando preferencia a aquellos con más recursos computacionales). De forma adicional, el maestro realiza reservaciones de esclavos y notifica a los clientes cuando estos se encuentran disponibles. Por último, el maestro genera informes indicando el estado actual del sistema, ya sea como texto en una consola o en forma de un documento en formato HTML.

2.4.3 Características especiales

Q^2ADPZ cuenta con las siguientes características especiales:

- Los clientes realizan sus labores de dos formas. Una forma es por medio de la ejecución controlada de binarios ejecutables por sí mismos. La otra forma es por medio de llamadas a funciones predeterminadas en bibliotecas de carga dinámica
- Las dos aplicaciones que forman el lado servidor pueden ejecutarse en equipos diferentes
- Se puede reservar un grupo de clientes para asignarles tareas exclusivas posteriormente
- El cliente identifica el hardware y software donde se ejecuta para que se le asignen trabajos acordes a su plataforma
- El cliente se ejecuta como un servicio
- Se puede monitorear el progreso de un proyecto vía Internet
- Se usan firmas digitales sobre los ejecutables transmitidos del servidor al cliente
- Sólo requiere como software adicional de un servidor Web para su funcionamiento

2.4.4 Seguridad y tolerancia a fallos

Los mecanismos de seguridad de Q²ADPZ se enfocan a la integridad como sistema, esto quiere decir que es muy difícil obtener acceso a alguna de las computadoras que forman el sistema y utilizarlas para otros fines distintos a los planeados. Para esto se diseñó el sistema teniendo en mente dos especificaciones:

- Sólo los usuarios registrados son capaces de enviar código ejecutable a las máquinas que funcionan como esclavos.
- El código esclavo tiene acceso limitado al ambiente anfitrión

Para cumplir con la primera especificación, todos los comandos de las aplicaciones cliente a la aplicación maestro se firman con una pareja formada por nombre de usuario y contraseña (ambos cifrados), así sólo los usuarios registrados previamente pueden enviar trabajos. Una lista con los nombres de usuario y contraseña se mantiene cifrada en el equipo que funciona como maestro. En el caso de la comunicación maestro a esclavo, se hace uso del cifrado de clave pública. Así, todos los datos que viajan del maestro al esclavo se cifran con la clave privada del maestro y el esclavo las descifra por medio de la clave pública (del maestro).
Es importante señalar que el flujo de datos del esclavo al maestro y del maestro al cliente no se cifra ni se firma

El mecanismo para cumplir la segunda especificación depende totalmente del sistema operativo anfitrión del esclavo. En este caso, se crea un usuario denominado esclavo con accesos limitados al sistema de archivos, es decir, está prohibida la escritura fuera de un directorio especial para descargar archivos.

Resumiendo, estos mecanismos de seguridad tienen las siguientes ventajas:

- Los esclavos no ejecutarán código no autorizado por el maestro
- Se protege el ambiente anfitrión del esclavo contra código dañino (intencional o accidental) creado por usuarios legítimos.

En cuanto a desventajas se pueden mencionar las siguientes:

- Sobrecarga de operación debido al cifrado/descifrado.
- Los accesos limitados a la aplicación cliente sólo se pueden lograr en sistemas de archivos tipo UNIX, por lo que es inútil en el caso de FAT o FAT32(comunes en S.O. Windows)

- Sólo se protegen los comandos de ordenes entre las aplicaciones, los archivos binarios como ejecutables no se encuentran cifrados, por lo que son propensos a copiado (mas no ejecución ya que la orden de ejecución sí es cifrada)

Respecto a la tolerancia a fallos, el sistema reasigna labores cuando detecta que un esclavo se encuentra bloqueado o fuera de servicio, esto de forma automática. Sin embargo, no existen mecanismos para reponerse de fallas intencionales, por lo que deben implementarse, en especial en ambientes sin control (como Internet)

2.4.5 Creación de una aplicación

1.- Debido a que actualmente no se distribuyen binarios compilados, es necesario descargar el código fuente de la página Web de desarrollo. De forma adicional se deben de descargar las bibliotecas libcurl, libcrypto y bzip2 si es que no se encuentran disponibles en el sistema operativo (las últimas distribuciones de GNU/Linux las incluyen)

2.- Se procede a la compilación del código fuente. Como es común en estos casos, el código se distribuye junto con archivos que automatizan la compilación (archivos make), tanto para Unix como Windows.

3.- Al final del proceso de compilación, si éste fue correcto, se obtendrán los siguientes archivos binarios, los cuales forman parte de alguna aplicación:

aplicación esclavo: aqadpz_slave (ejecutable principal)
libslv-app.so (biblioteca encargada de iniciar los programas usuarios esclavos ejecutables o interpretados)

aplicación maestro: qadpz_master (ejecutable principal)
qadpz_admin (utilidad para la administración de usuarios, claves de seguridad y actualización de aplicaciones esclavo)

aplicación cliente: qadpz_run (ejecutable principal, usado para iniciar y controlar los trabajos descritos en los archivos de proyecto en formato XML)
libcli_serv.a (biblioteca que proporciona las llamadas API cliente)

4.- Junto con el código fuente se ponen a disposición archivos de configuración necesarios para cada una de las aplicaciones. Estos archivos se denominan "master.cfg", "slave.cfg" y "client.cfg" los cuales se usan en la aplicación maestro, esclavo y cliente, respectivamente. Salvo algunos datos especiales, sólo basta con indicar la dirección IP del maestro en todos los archivos de configuración.

5.- Utilizando el ejecutable qadpz_admin se crea una pareja de clave pública y privada. Al final del proceso se obtendrán dos archivos denominados "pubkey" y "privkey".

6.- Se crea una lista de usuarios autorizados por medio de la aplicación qadpz_admin. Esta lista se denomina "users.txt".

7.-Se empaca, distribuye y se configuran con permisos adecuados la aplicación esclavo(formada por los archivos "qadpz_slave", "libslv-app.so" y "slave.cfg") a los equipos que funcionaran como esclavos. De igual forma para la aplicación cliente (archivos "qadpz_run", "client.cfg" y "pubkey") y la aplicación maestro ("qadpz_master", "qadpz_admin", "master.cfg", "users.txt" y "privkey").

8.- En el equipo que funcionará como maestro se le agregan unos archivos cgi (contenidos en la distribución del código fuente) al servidor Apache. Estos archivos se ocupan de las cargas y descargas de archivos.

9.- Se ejecuta el binario qapz_master. Con esto el sistema como conjunto está completo, por lo que sólo basta que las aplicaciones clientes ejecutadas por los usuarios pidan la solución de tareas previamente creadas (formadas por los programas usuarios esclavos, archivos de entrada y configuración)

2.4.6 Evaluación

En la evaluación de la herramienta MODELS@home se había hecho el comentario de la importancia de mejorar el desempeño de una aplicación secuencial al ejecutarla en un ambiente de cómputo distribuido. Pues bien, en Q²ADPZ se encuentra este tipo de

propiedad, por lo que es posible la mejora del desempeño en ejecutables de los que no se cuenta el código fuente. Todavía mejor, se pueden obtener incrementos en el desempeño mayores cuando se tiene a la mano el código fuente del algoritmo de solución que se desea implementar, al compilarlo como biblioteca de carga dinámica y realizar llamadas API del sistema.

Esta facilidad respecto a la implementación del algoritmo de solución permite que existan varios niveles de uso del sistema, del básico, en el cual se utilizan ejecutables ya hechos (ideal para aquellos usuarios con pocos conocimientos de programación), hasta el avanzado, donde el usuario experimentado crea sus bibliotecas de alto desempeño.

Un aspecto muy importante de Q^2ADPZ es que como sistema, se apoya en software profesional, como el servidor de Web Apache, para mejorar su escalabilidad y desempeño.

Sin embargo, existen ciertos detalles, que si bien no son significativos en ciertos ambientes, en otros parecerían imperdonables, como la carencia del uso de administrador de base de datos o que la aplicación esclavo se instale como servicio y no informe al dueño del equipo de su actividad y finalmente, la omisión de un mecanismo de seguridad para solventar fallas de tipo intencionales.

Resumiendo, aunque esta herramienta cuenta con varias fallas, las opciones que ofrece, hacen que esta herramienta sea un sólido candidato a tomar en cuenta para implementar proyectos de cómputo distribuido, tanto en ambientes tipo LAN como un red heterogénea como es Internet.

2.5 BOINC

La Infraestructura Abierta para Cómputo de Red de Berkeley o BOINC (Berkeley Open Infraestructure for Network Computing) [2] es una plataforma para el cómputo distribuido usando recursos computacionales aportados de forma voluntaria.

Esta herramienta está siendo desarrollada actualmente en la Universidad de Berkeley. De forma adicional, se puede comentar que una versión mejorada del proyecto SETI@home se está preparando utilizando esta herramienta.

2.5.1 Información técnica

Los detalles técnicos más relevantes se resumen a continuación:

Sistemas operativos para la ejecución de la aplicación maestro:
Windows 9x/Me/2000/XP, GNU/Linux, Mac OS X

Sistemas operativos para ejecución la aplicación trabajador: Windows 9x/Me/2000/XP, GNU/Linux, Mac OS X

Lenguajes de programación que soporta: C,C++

Bibliotecas usadas: RSAEuro, libmysql, libz

Licencia: Mozilla Public License version 1.0

Software adicional: servidor Apache, administrador de base de datos Mysql, intérprete PHP, ambiente Cygwin si se desea compilar bajo Windows.

2.5.2 Arquitectura

Aunque BOINC tiene más de dos aplicaciones, éstas se pueden agrupar en dos: cliente y servidor, siendo el cliente el más sencillo.

El cliente está formado por el núcleo cliente y la aplicación cliente. El núcleo cliente proporciona la funcionalidad necesaria para interactuar con el equipo donde se ejecuta y con el servidor. Por medio de llamadas a la Interfaz de Programa de Aplicación (API por sus siglas en inglés) la aplicación cliente interactúa con el núcleo cliente, a la vez que implementa el algoritmo de solución que se desea. En el caso de los proyectos realizados con BOINC, el algoritmo de solución se implementa por medio de un archivo ejecutable, es decir, no se pueden usar programas interpretados ni bibliotecas de carga dinámica. En conjunto, el cliente tiene las siguientes funciones:

1) Obtener unidades de trabajo cuando sea necesario

2) Resolver los requerimientos de archivos de las unidades de trabajo por medio de la descarga de estos de un servidor Web.
3) Registrar los recursos computacionales con los que cuenta el equipo donde se ejecuta.
4) Permanecer en estado latente en el equipo donde se ejecuta para realizar las unidades de trabajo cuando sea conveniente.
5) Llevar un registro del tiempo invertido en realizar una unidad de trabajo.
6) Enviar los resultados que se obtengan al servidor.

Como se puede apreciar, es el cliente quien realiza las labores que le asigna el servidor y por lo tanto se procura que se ejecute en la mayor cantidad de equipos posibles.

El servidor se puede dividir en dos: el complejo BOINC y el extremo posterior del proyecto.
El extremo posterior del proyecto esta formado por varias aplicaciones denominadas generador de trabajo, verificador y asimilador, apoyadas por el servidor de datos. El generador de trabajo tiene la función de crear unidades de trabajo y archivos de entrada para los clientes; el verificador se encarga de comparar resultados redundantes de una misma labor y determinar cual es el correcto; por último el asimilador se encarga de manejar los resultados correctos que se obtengan, registrarlos y ordenarlos en la base de datos. Cabe aclarar que estas aplicaciones deben ser implementadas por los desarrolladores del proyecto siguiendo ciertos lineamientos propuestos por los creadores de BOINC. Finalmente el servidor de datos consiste en un servidor HTTP capaz de soportar programas CGI.
El complejo BOINC tiene como función principal la distribución de los datos. Para poder realizar sus labores se encuentra divido en varios componentes como uno o más servidores planificadores, los cuales se encargan de la comunicación entre equipos; una base de datos relacional, la cual almacena información relacionada con los trabajos, resultados y participantes; bibliotecas y programas de utilería, los cuales permiten al extremo posterior del proyecto comunicarse con el complejo BOINC; interfaces Web para los participantes y desarrolladores. El diagrama de esta arquitectura se muestra en la figura 2.5

Hasta el momento sólo se han mencionado las aplicaciones que forman la arquitectura. Faltan de mencionar los datos que circulan entre las aplicaciones cliente y servidor. Estos básicamente son archivos en formato XML, ejecutables y otros archivos binarios.
Los archivos en formato XML sirven para comunicarse entre el cliente y el servidor. Estos se denominan lotes, unidades de trabajo y resultados. Un lote esta formado por varias

unidades de trabajo. Una unidad de trabajo contiene la información necesaria para que un cliente pueda desempeñar su labor. En una unidad de trabajo se indican: argumentos de línea de comandos, ejecutable, tiempo máximo para realizar el trabajo y archivos binarios de entrada para el ejecutable. En un resultado se indica la unidad de trabajo a la que pertenece, el archivo o archivos binarios de salida y el tiempo invertido.

2.5.3 Características especiales

BOINC cuenta con las siguientes características especiales:

En el lado cliente:

- Los clientes realizan sus labores por medio de la ejecución controlada de archivos. binarios ejecutables por sí mismos.

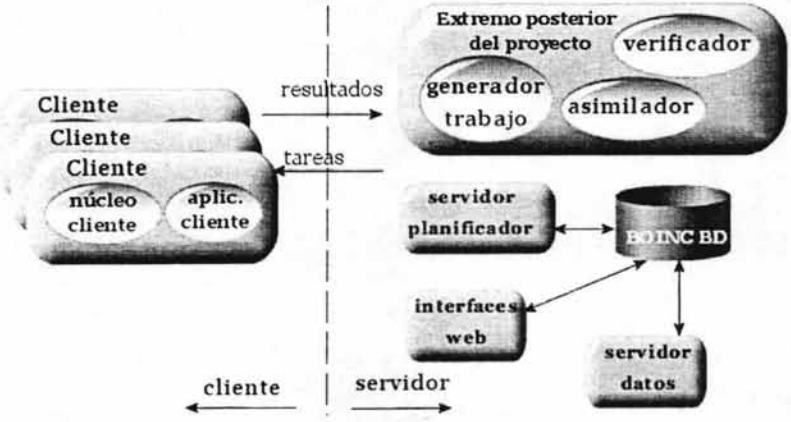

Figura 2.5 Arquitectura de BOINC

- Los clientes trabajan como protectores de pantalla o servicios de fondo.
- El protector de pantalla puede ser definido por el desarrollador y ser diferente por cada tarea.
- Los clientes cuentan con un mecanismo para evitar quedarse sin trabajo cuando no exista conexión a red.
- Se lleva un registro del tiempo de procesador usado en una tarea, con posibles fines de retribución económica o en especie.
- Opción para el dueño del equipo de limitar los recursos donados.

- Uso de cifrado de clave pública con fines de seguridad.
- Opción de guardar avances en una tarea que no ha sido completada, para retomarlos después.

En el lado servidor:

- Apoyo en software especializado para transferencia de archivos de todo tipo.
- Uso de un administrador de base de datos para funciones de registro de los archivos transferidos, cuentas de usuarios, etc.
- Manejo de clientes individuales o grupos.
- Capacidad para distribuir la carga por medio del uso de varios servidores.
- Uso de cifrado de clave pública con fines de seguridad.
- Redistribución de las tareas en caso de estancamientos.
- Capacidad para manejar tareas que toman mucho tiempo y espacio en memoria.
- No es necesario que las distintas aplicaciones que forman el servidor se ejecuten en el mismo equipo.
- Programas interpretados para simular varias condiciones en el proyecto y así medir su desempeño.

2.5.4 Seguridad y tolerancia a fallos

Como otras herramientas similares, BOINC hace uso del cifrado de clave pública. Para esto, se crean una pareja de clave pública/privada. La clave pública se proporciona a todos los clientes que intervienen en el proyecto, mientras que la clave privada se mantiene en el servidor. Todos los archivos que viajan del servidor al cliente son firmados digitalmente utilizando la clave privada, de forma que un cliente no aceptará un archivo que no tenga la firma adecuada.

Para mejorar la seguridad, la pareja de clave pública/privada es actualizada constantemente. El procedimiento de actualización consiste en crear una pareja nueva de clave pública/privada. Entonces se procede a enviar la nueva clave pública a los clientes, está firmada con la antigua clave privada; una vez verificada la firma, el cliente sólo trabajara con la nueva clave pública hasta que ocurra una nueva actualización.

Debido a que el servidor de datos puede ejecutarse en un equipo distinto a donde se ejecutan las demás aplicaciones que forman el servidor, existe un mecanismo especial para ingresar

nuevos archivos al servidor de datos. Este mecanismo también se basa en cifrado de clave pública/privada, con fines de autenticar a los usuarios válidos del proyecto.

Resumiendo, estos mecanismos de seguridad tienen las siguientes ventajas:

- Los clientes no ejecutarán código inautorizado por el servidor.
- Los archivos sólo viajan firmados, no cifrados, evitando sobrecarga de operaciones.
- Mejora en la seguridad por medio de actualización constante de claves.

En cuanto a desventajas se pueden mencionar las siguientes:

- Al ser solamente firmados los archivos, son propensos a ser copiados.
- No hay protección contra código malicioso autorizado por el servidor

Respecto a la tolerancia a fallos, existen dos elementos por validar en un proyecto: los resultados y el crédito por uso de CPU de un cliente. En ambos casos, la redundancia es empleado como mecanismo de tolerancia a fallos. Para ello, se distribuyen las tareas a más de un cliente y cuando un mismo resultado sea entregado por la mayoría de los clientes, es aceptado y el crédito por tiempo de uso de CPU que se le otorga a los clientes es el de aquel cliente que indique el menor tiempo.

2.5.5 Creación de una aplicación

Nota.- Debido a que pueden utilizarse múltiples equipos para formar al servidor, el proceso que aquí se describe es una instalación del servidor en un mismo equipo. Debe asegurarse que el equipo cuente con el servidor MySQL y Apache instalados.

1.- Debido a que actualmente no se distribuyen binarios compilados, es necesario descargar el código fuente de la página Web de desarrollo. De forma adicional se deben de descargar las bibliotecas libmysql y libz si es que no se encuentran disponibles en el sistema operativo (las últimas distribuciones de GNU/Linux las incluyen)

2.- Se procede a la compilación del código fuente. Como es común en estos casos, el código se distribuye junto con archivos que automatizan la compilación (archivos make). Si se desea compilar en plataforma Windows, es necesario utilizar el ambiente Cygwin.

3.- Al final del proceso de compilación, si éste fue correcto, se obtendrán un número considerable de archivos binarios, los cuales no se indicarán aquí. Basta saber que se dividen en bibliotecas de carga dinámica y binarios ejecutables y que una parte de estos conforman al cliente y otra al servidor.

4.- Debido a que el resto de la configuración del servidor es un tanto compleja, se proporciona una serie de programas interpretados en lenguaje PHP que facilitan esta labor. Se debe indicar por medio de variables de ambiente algunos parámetros para los programas interpretados. Estas variables son la ruta para crear un nuevo directorio de proyecto, un nombre de usuario, el directorio con el código fuente BOINC, la dirección URL del enlace simbólico al directorio que contiene los programas CGI que se utilizarán, la dirección URL al enlace simbólico al directorio que contendrá archivos html de interfaz Web y por último un número de 32 bits que indique una posición de memoria compartida no ocupada.

5.- A continuación se procede a ejecutar el programa interpretado "make_project.php". Este programa realiza las siguientes labores:

- Crea el directorio y subdirectorios para un nuevo proyecto.
- Crea la pareja de clave pública/privada que se utilizarán en el proyecto.
- Crea una nueva base de datos MySQL y agrega algunos registros iniciales a las tablas que la forman.
- Copia los programas propios del servidor (aquellos compilados en el paso 2) más algunos archivos PHP distribuidos con el código fuente a sus nuevos directorios.
- Configura permisos de archivos.
- Crea enlaces simbólicos de archivos html y programas CGI.

6.-Es necesario editar el archivo de configuración del servidor Apache, esto con fines de habilitar la ejecución de programas CGI y las opciones para subir archivos al servidor, debido a que el programa interpretado anterior no lo hace.

7.-Se empacan y distribuye a cuantos equipos sea posible los archivos que conforman el cliente. No olvidar incluir la clave pública, si es que se quiere hacer uso de este mecanismo de seguridad.

8.- El servidor se encuentra listo para ejecutarse. Sólo basta desarrollar el extremo posterior del proyecto, crear las tareas (formadas por los programas ejecutables, los archivos de

entrada y configuración) e ingresarlas al servidor de datos para que éste se encargue de distribuirlas y obtener los resultados.

2.5.6 Evaluación

Los proyectos creados con BOINC tienen cierta ventaja sobre los demás. Una de estas ventajas es un buen desempeño bajo una gran carga de clientes. Para lograr esto, los creadores de BOINC decidieron hacer uso de aplicaciones de software libre profesional como es el servidor Apache, MySQL y PHP. Otra ventaja consiste en que el proyecto resulta atractivo a los posibles donadores de recursos al proporcionar características como despliegues gráficos tipo protector de pantalla, créditos por el tiempo donado, creación de grupos de colaboración y una misma cuenta de registro, esta última independiente del equipo de ejecución.
En general, en BOINC se encuentran características profesionales, muchas de ellas exclusivas de proyectos de cómputo distribuido creados desde cero, lo cual lo hace llamativo a los posibles desarrolladores. Otro punto importante a destacar es que esta herramienta es de las pocas que integran mecanismos de tolerancia a fallas intencionales.

La integración de aspectos profesionales tiene un costo, el cual radica en una complejidad de administración del servidor mayor. De hecho, ésta es una de las herramientas más difíciles de usar. Se necesita de una inversión de tiempo mayor antes de empezar a desarrollar un nuevo proyecto. Además, es extraño que no se ofrezca cifrado para los archivos que viajan entre el cliente y servidor, siendo que otras herramientas menos adornadas lo integran por omisión.

Ya para finalizar, se puede comentar que, a pesar de su complejidad de instalación y uso, el proyecto generado con esta herramienta es de buena calidad, en especial para un ambiente de red como Internet y ofrece características atractivas, tanto para los posibles donadores como a los administradores del proyecto. No en vano la siguiente versión de SETI@home se está desarrollando utilizando esta herramienta.

En el siguiente cuadro se muestra la comparación de los aspectos más relevantes de todas las herramientas presentadas en este capítulo:

Tabla comparativa de herramientas de cómputo distribuido

Característica	FIDA	MODELS@home	Q2ADPZ	BOINC
S.O. Maestro	Windows 98, ME, NT, 2000,XP, GNU/Linux, MAC OS X	Windows 98, ME, NT, 2000,XP, GNU/Linux	Windows 98, ME, 2000,XP, GNU/Linux, FreeBSD, SunOS, IRIX64	Windows 98, ME, 2000,XP, GNU/Linux, MAC OS X
S.O de trabajadores	Windows 98, ME, NT, 2000,XP, GNU/Linux, MAC OS X	Windows 98, ME, NT, 2000,XP, GNU/Linux	Windows 98, ME, 2000,XP, GNU/Linux, FreeBSD, SunOS, IRIX64	Windows 98, ME, 2000,XP, GNU/Linux, MAC OS X
Reasignación de tareas en sistemas colapsados	No	Si	Si	Si
Tolerancia a trabajadores maliciosos	No	No	No	Si
Cifrado de datos en red	No	No	No	No
Autenticación de la información enviada por el maestro	Si	No	Si	Si
Software adicional necesario	Ninguno	Ninguno	Servidor Web Apache	Servidor Web Apache con PHP, DBMS MySQL,
Distribución de las funciones del maestro en equipos distintos	No	Si	Si	Si
Adaptación de código fuente de aplicaciones para trabajadores	Si	No	No	No
Identificación de recursos del trabajador	No	No	Si	Si

2.6 CONCLUSIONES

En este capítulo se presentó el estudio de una selección de infraestructuras disponibles, para la implementación de proyectos de cómputo distribuido. El estudio de éstas se realizo bajo criterios uniformes para así obtener un punto de comparación. Sin embargo, cada infraestructura tiene una orientación determinada y de ahí el origen de sus ventajas y desventajas. El conocimiento adecuado de cualquier infraestructura permite su mejor aprovechamiento y así, determinar si conviene o no a los intereses propios.

Con el estudio de éstas, se obtuvo información muy útil como guía para el diseño e implementación de Éxigus, tal como se puede notar en el siguiente capítulo.

3. DISEÑO E IMPLEMENTACIÓN DE ÉXIGUS

En el primer capítulo se presentó una introducción a la teoría básica en la cual se sustentan los proyectos de cómputo distribuido. En el capítulo 2 se mostró una selección de las herramientas actuales para crear proyectos de cómputo distribuido. Utilizando los conocimientos adquiridos en ambos capítulos es posible crear una nueva herramienta competitiva para el desarrollo de proyectos de cómputo distribuido, denominada Éxigus[9]. Sólo falta señalar que las pruebas de esta herramienta, una vez concluido su desarrollo, se presentan en un capítulo posterior.

3.1 OBJETIVOS DE LA INFRAESTRUCTURA ÉXIGUS

Éxigus está orientado al cumplimiento cuatro objetivos: flexibilidad, escalabilidad, seguridad y tolerancia a fallos. A continuación se describirán a mayor detalle cada uno de ellos:

- Flexibilidad se entiende como la capacidad de configurar fácilmente varios parámetros de la infraestructura, y a sí lograr una integración sencilla con aplicaciones existentes. Un ejemplo de esto se encuentra en el hecho de que usando Éxigus, no es necesario

[9] Las siglas de Éxigus carecen de significado en particular, sin embargo tienen similitud con la palabra latina *exigŭus*, que en español significa exiguo, de tal forma que se puede pensar en el nombre como un recordatorio de uno de los principales problemas que se desean solucionar en el cómputo distribuido: la escasez de recursos computacionales.

recompilar una aplicación existente secuencial, para cambiar su comportamiento a una de carácter distribuido. Además, el formato de la especificación de tareas puede ser tan sencillo o tan completo como el usuario lo desee, pudiendo integrar subtareas para un mejor control de la ejecución.

- Escalabilidad se entiende como la capacidad de poder distribuir toda la carga involucrada en la administración de los trabajadores, usando varios equipos físicos, en el momento que se considere conveniente. Los beneficios se traducen en el manejo de una mayor cantidad de trabajadores y una mayor capacidad de cómputo. Un ejemplo de esto se encuentra en la partición del componente Maestro de Éxigus en tres: un servidor de archivos, un administrador de bases de datos y un coordinador. Estos tres componentes pueden estar en un solo equipo, en dos, tres o más equipos físicos distintos, siempre y cuando tengan acceso a una red de comunicaciones.

- Seguridad se entiende como la capacidad de que el sistema en conjunto sólo pueda ser usado para lo que fue diseñado, protegiendo la confidencialidad, integridad y disponibilidad de la información. Sin embargo, hay que hacer notar que la seguridad total es muy difícil, de lograr, sino imposible. No obstante, diseñar un software sin tener la seguridad en mente desde un principio representa un riesgo muy alto, con consecuencias igualmente graves. Un ejemplo de un elemento de seguridad en Éxigus consiste en el uso de firmas digitales para autenticar el origen de las aplicaciones que ejecutan los trabajadores. De esta forma, aunque una aplicación sea modificada durante su tránsito del Maestro al trabajador, el trabajador puede darse cuenta de ello y evitar la ejecución de código malicioso.

- Tolerancia a fallos se entiende como la capacidad tanto del Maestro como del trabajador de poder recuperarse de situaciones que pudieran impedir un desempeño exitoso. Por ejemplo, todos los trabajos que se deben de realizar usando Éxigus son distribuidos y controlados usando una cola circular, en la que basándose en un orden arbitrario, cuando un trabajador solicita un trabajo, se le asigna el siguiente al último asignado, lo cual garantiza que mientras exista un trabajador disponible, todos los trabajos ser resolverán.

Hay que hacer notar que los objetivos anteriores están sujetos a cierta interpretación personal, por que bajo algunos criterios, su cumplimiento pudiera parecer parcial. Sin embargo, todos los objetivos antes presentados son parte de un objetivo mayor, el cual es desarrollar un producto útil y de calidad.

3.2 CONSIDERACIONES PARA EL DISEÑO DE LA ARQUITECTURA

Para el diseño básico de la arquitectura se puede partir de las ideas fundamentales que se han presentado a lo largo del trabajo, las cuales se resumen en el siguiente esquema básico:

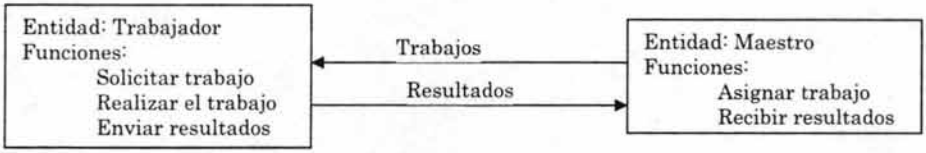

figura 3.1 Arquitectura y funcionalidad básica de Éxigus

La relación de trabajadores y maestros se ilustran a continuación, donde se puede observar como un Maestro tiene a su cargo varios trabajadores, de los cuales atiende sus solicitudes de trabajos y recibe sus resultados:

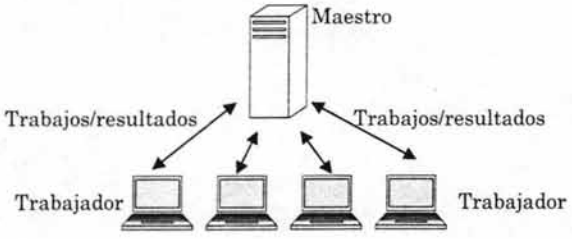

figura 3.2 Relación de número de trabajadores y maestros

Con base en como se definan los trabajos, los resultados y las funciones de asignación-recepción de trabajos, envío y recepción de resultados, será como la arquitectura tomará su forma final.

3.2.1 Requisitos para realizar un trabajo

Un trabajo contiene las indicaciones que debe seguir un trabajador y que son entendibles para él. Para que un trabajador pueda realizar un trabajo se necesita que se cumplan tres requisitos:

1) el trabajador debe saber que es lo que tiene que hacer
2) el trabajador debe saber como se hace lo que tiene que hacer
3) el trabajador debe tener los implementos necesarios para realizar su labor.

Para ilustrar estos requisitos, se puede hacer uso de una analogía. Imagínese a un empleado en un ambiente laboral típico de oficina. Cuando el oficinista se percata de que ha concluido sus labores anteriores, se acerca con su jefe para que le asigne más trabajo. Supóngase que el jefe le indica "...realice un informe de ventas del 2° trimestre del año anterior...". En este momento el empleado sabe que es lo que tiene que hacer, sin embargo, para que pueda cumplir con el encargo, necesita saber cómo se elabora un informe y también debe tener acceso al historial de ventas del año anterior.

En el mundo del cómputo distribuido, estos requisitos funcionan prácticamente igual. Supóngase que una entidad trabajador solicita una labor la cual consiste en representar[10] una escena de un modelo tridimensional que usa texturas personalizadas. El trabajador sabe que es lo que tiene que hacer (obtener una imagen), pero debe saber como se hace la representación de un modelo y una vez que sepa esto tiene que tener acceso a las texturas personalizadas para obtener la imagen final.

En el mundo real una persona toma cursos de capacitación para aprender a realizar una nueva labor muy concreta. Pues si bien no se puede darle cursos a una computadora, es posible proporcionarle una lista de instrucciones muy concretas que debe seguir para que pueda realizar lo que se le pide. Esta lista de instrucciones se conoce como un programa.

Regresando al problema de la representación de una imagen, es necesario proporcionarle al trabajador, aparte del modelo tridimensional y sus texturas, el programa para convertir los datos anteriores en una imagen. Este programa puede estar integrado en el trabajador o se

---

[10] En inglés, rendering

puede acceder a él de forma externa. En el primer caso se dice el programa está integrado como un procedimiento estático. En el otro caso, el programa ajeno debe tomar la forma de un programa ejecutable por sí mismo[11] o de un procedimiento de biblioteca de enlace dinámico.

Los procedimientos estáticos tienen la ventaja de que son más rápidos en su ejecución en el sentido de que su llamada es directa e instantánea, ya que se cargan junto con el programa con el que van incluidos. En cambio, los programas ejecutables por sí mismos o los procedimientos de biblioteca de enlace dinámico sufren de tiempos de espera en lo que se cargan a memoria. Sin embargo, las ventajas de estos últimos radican en que se puede modificar su funcionalidad sin necesidad de compilar de nuevo el programa que los llama, cosa que es imposible con los procedimientos estáticos.

Cuando se tiene una idea concreta de la clase de trabajos que se pretende que haga un trabajador se prefieren los procedimientos estáticos. Sin embargo, ya que Éxigus es una infraestructura para proyectos de cómputo distribuido de cualquier tipo, se utilizarán programas ejecutables por si mismos como forma de "enseñarle" al trabajador como realizar una labor. Esto tiene la ventaja adicional de que se pueden utilizar programas ya existentes de los cuales no se necesita conocer su código fuente, lo cual no podría pasar si se quisiera implementar determinada funcionalidad en forma de procedimiento estático.

Generalmente un programa requiere de cierta entrada para generar una salida. En el caso del problema de la representación de una escena, se requiere como entrada una secuencia de bytes con la definición del modelo tridimensional con el que se quiere trabajar y las secuencias de bytes de las texturas personalizadas. La salida que se genera es una imagen en cualquier formato (.jpg, .bmp, .png, etc.)
A veces parte de la entrada consiste no en datos sobre los cuales trabajar, sino más bien una serie de indicaciones cortas o modificadores. Por ejemplo, se le puede indicar al programa de representación que genere como salida una imagen específicamente en formato .jpg, a diferencia de un formato .bmp que podría generar por omisión.

---

[11] En inglés, standalone executable. Esta clase de programas no requiere de ningún programa adicional para ejecutarse, como podría ser un intérprete.

Resumiendo, en Éxigus para que un trabajador pueda realizar cualquier trabajo, se necesita de un programa ejecutable con las instrucciones de cómo debe realizar algo, los datos de entrada sobre los cuales trabajar y una serie de indicaciones o modificadores. En algunas ocasiones los datos de entrada o los modificadores o ambos pueden omitirse, aunque esta última clase de omisión no es muy común.

3.2.2 Aspectos a considerar en los trabajos

Ahora que se conocen los elementos necesarios para realizar un trabajo es el momento de definir el formato concreto de un trabajo. Un trabajo está formado por un programa ejecutable, los datos de entrada sobre los cuales trabajar y la lista de modificadores que alteran la ejecución del programa. Esta lista de modificadores se conoce como la línea de comandos o argumentos que se le pasan a un programa antes de iniciar su ejecución.

Debido a que es necesario que estos elementos (ejecutable, datos entrada, línea de comandos) estén disponibles al momento de realizar un trabajo, lo mejor es que sean empacados en un solo archivo y que se desempaquen cuando se necesiten. En Éxigus se utiliza el formato Zip para empacar todos los datos que conforman un trabajo.

Un ejemplo de un trabajo podría ser el hipotético archivo comprimido en formato Zip "`trab_rep-001.zip`"

Este archivo comprimido contendría a su vez los siguientes archivos:

Nombre de archivo	Comentario
`render.exe`	Archivo ejecutable utilizado para representar una escena
`House_Scene.3d`	Escena lista para representar
`wood.txtr`	Textura usada para la representación
`trabajo.cmd`	Línea de comandos para ejecutar el programa `render.exe`. Es un archivo de texto plano con una sola línea con un contenido similar al siguiente: `render.exe House_Scene.3d -t wood.txtr -format jpg -out house.jpg`

Cuando un trabador quiera realizar el trabajo `trab_rep-001.zip`, descomprimirá este archivo a un directorio y lo ejecutará de acuerdo con las instrucciones en `trabajo.cmd` para que al final del proceso se obtenga como resultado una imagen en formato JPG denominada `house.jpg`

Un detalle importante es que el trabajador debe saber cual es el archivo con la línea de comandos. Este archivo puede tener un nombre o una extensión determinados para así distinguirse de los demás archivos. En la infraestructura Éxigus, el archivo que contiene estas instrucciones se identifica como un archivo que tiene el mismo nombre (sin la extensión) que el archivo comprimido que lo contiene, más una extensión .xml. De esta forma, para el archivo `trabajo01.zip`, el archivo con la línea de comandos debe llamarse `trabajo01.xml`

En algunas ocasiones se necesitan ejecutar varios programas en secuencia para obtener un solo resultado. Por ejemplo, para generar un programa ejecutable a partir de un archivo de código fuente en lenguaje C se necesitan tres programas: un preprocesador, un compilador y un enlazador. Aunque estos pueden incluirse en un solo programa, lo más común es que estén separados y que se llamen conforme avanza el proceso de la creación de un nuevo ejecutable.
Así como está definido un trabajo actualmente, sólo se puede ejecutar un programa por trabajo. Para cambiar esta situación y permitir ejecutar varios programas uno o más veces sólo se necesitan incluir los ejecutables respectivos con sus respectivos datos de entrada y permitir que el archivo con la línea de comandos incluya una línea de texto por cada programa que desea ejecutar.

Tomando en cuenta todo lo anterior, un trabajo es un archivo comprimido en formato Zip que contiene tentativamente:

- 1 ó más programas ejecutables
- 0 ó más archivos de entrada para los ejecutables
- 1 archivo indicando la secuencia y parámetros de ejecución de los programas

Se dice que el contenido es tentativo debido a un detalle relacionado con los programas ejecutables. En un proyecto de cómputo distribuido lo más común es que los datos de entrada sean la clase de archivos que más varían, mientras que los ejecutables no. Por ejemplo, si se quiere representar una película completa, lo común es que se distribuyan entre los trabajadores diferentes escenas de la película, mientras que el programa para hacerlo sea el mismo.
Por lo tanto, es un desperdicio en tiempo de transmisión de archivos y espacio en disco duro, el incluir en el trabajo los ejecutables junto con los demás archivos que forman el trabajo. Lo

más recomendable es que estos ejecutables se obtenga de una sola vez por un medio distinto al de obtener un trabajo.

3.2.3 Aspectos a considerar para los resultados

Debido a que se pueden ejecutar varios programas, y generar por tanto, varios archivos en el transcurso de la realización de un trabajo, es necesario definir cuáles de esos archivos son los que se considerarán como resultados que se enviarán al maestro que asignó el trabajo. Es necesario que alguna parte del trabajo se indique cuáles son los archivos que se considerarán como resultados. Para ello puede anexarse un archivo de texto plano denominado por ejemplo, "resultados.txt" en donde se haga un listado de aquellos que se enviarán al maestro.

¿Qué pasaría si un programa falla en su ejecución y no genera el archivo de resultado que debiera?, y ¿cómo se sabría el motivo de su falla? Conocer esto es muy importante para tomar las medidas adecuadas.

Cuando un programa ejecutable termina su ejecución regresa un valor numérico al programa que lo llamó. Este valor indica el motivo de su término. Por convención general un valor de regreso de 0 indica que el programa tuvo una terminación normal, mientras que un valor distinto indica una terminación anormal y su motivo. Por ejemplo, el programa `zip.exe` regresa un valor de 7 cuando no hay espacio suficiente en disco duro para crear un nuevo archivo y un 3 para indicar un argumento de línea de comandos erróneo.

Ya que no todos los programas siguen las mismas convenciones, es necesario indicar en el trabajo cuales son los códigos de salida correctos para un programa. De nueva cuenta se puede hacer con un archivo de texto plano denominado, por ejemplo, "códigos de salida.txt" en donde haya tantas líneas como programas indicando el nombre del ejecutable y su código de salida correcto.

Una vez conociendo los archivos que se consideran como resultados, lo único que falta es empacarlos en un solo archivo, y en el caso de que existiera una falla en la ejecución de un programa, indicarlo por medio de un archivo de informe. Este archivo contendría el nombre del ejecutable que falló y su código de salida.

3.2.4 Solicitud y asignación de trabajos

La forma más simple de solicitar y asignar trabajos entre un trabajador y un maestro, consiste en que el trabajador inicie una conexión de red en un puerto predeterminado con la entidad maestro y posteriormente, se realice la transferencia de los bytes que conforman el trabajo (el archivo empaquetado). En el caso de que no existiera trabajo por asignar, el maestro puede enviar una secuencia de bytes con un significado especial, por ejemplo "NO_HAY_TRABAJO". Esto se ilustra mejor en las siguientes figuras:

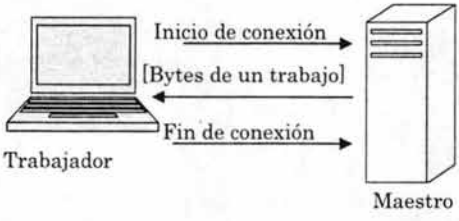

figura 3.3 Solicitud-asignación de trabajo cuando hay

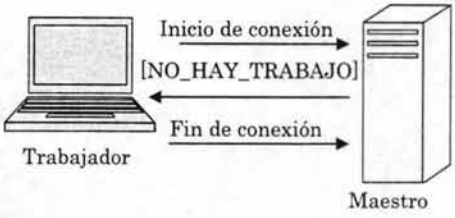

figura 3.4 Solicitud-asignación de trabajo cuando no hay

Existen varias situaciones que hay que considerar antes de decidir la implementación final de la funcionalidad de solicitud y asignación de trabajos, dos inmediatas son:

1) Las conexiones pueden terminar de forma abrupta en cualquier momento
2) El maestro tiene que atender solicitudes simultaneas de varios trabajadores

Una conexión entre el trabajador y el maestro puede terminar por diversos motivos como fallas en los medios de transmisión, en los equipos e incluso terminaciones explicitas por parte de usuarios. En trabajos pequeños no importa si termina la conexión, simplemente se descartan los bytes descargados y se realiza una nueva conexión. Sin embargo, en trabajos de mayor tamaño podría suceder que nunca se realizara de forma completa la transferencia de la información. Por lo tanto, es necesario implementar de alguna forma mecanismos que permitan reanudar la transmisión de información en caso de terminaciones prematuras.

Respecto a la atención de solicitudes simultaneas, es necesario hacer uso de técnicas de programación multihilo o multiproceso para lograr el objetivo. Básicamente existe siempre

un proceso o hilo[12] principal a la espera de alguna solicitud. Cuando se da esta solicitud se hace cualquiera de los dos casos:

1) se crea un proceso o hilo nuevo

2) se toma un proceso o hilo de un depósito especial denominado piscina

Este proceso o hilo creado o tomado se encarga de atender la solicitud actual, mientras que el proceso o hilo principal sigue a la espera de nuevas solicitudes. De esta forma puede atenderse a un gran número de trabajadores, limitados solamente por los recursos computacionales del equipo donde se ejecuta el maestro.

Implementar la funcionalidad necesaria para afrontar los posibles escenarios que se pudieran presentar no es rápido ni sencillo. Sin embargo, existen programas que están diseñados en especial para solventar gran parte de los problemas relacionados con la transmisión y recepción de grandes bloques de información: los servidores de archivos de red basados en protocolos como ftp, http o scp. Con un servidor de archivos se elimina gran parte de la complejidad de la entidad maestro, además de reducir los tiempos de desarrollo y pruebas.

Con el uso de un servidor de archivos hay que realizar un cambio en como se recibían los trabajos. Antes los trabajos se descargaban directamente al iniciar una conexión, si es que había disponibles. Ahora con el uso de un servidor de archivos, el maestro al detectar una solicitud de trabajo, no envía el trabajo directamente, sino que proporciona una liga o indicación de cómo descargar el trabajo del servidor de archivos.

3.2.5 Categorización y control de los trabajos

Dependiendo de la clase de los proyectos, pueden existir categorías en los trabajos. Estas categorías determinan los requerimientos mínimos de hardware para que un trabajo pueda realizarse exitosamente.

---

[12] Un proceso es una instancia de la ejecución de un programa Un hilo es una parte del programa de un proceso, la cual se ejecuta por separado, sin embargo, tiene acceso a todas las variables y datos del proceso que lo originó.

La idea de manejar categorías de trabajos se maneja en varios proyectos. Por ejemplo, en el proyecto GIMPS (Great Internet Mersenne Prime Search) los trabajos se dividen en tres categorías las cuales requieren de mayor a menor velocidad de procesamiento en el hardware del equipo donde se ejecuta el trabajador. Los trabajos de la primera categoría se encargan de buscar nuevos números primos Mersenne, mientras que en la segunda categoría se realizan pruebas de verificación y en la tercera factorizaciones de apoyo.

Esta categorización es muy importante ya que se optimiza la utilización de recursos al no asignar trabajos muy pesados de realizar a trabajadores con equipos lentos o al contrario, asignar labores muy sencillas a trabajadores que se ejecutan en equipos poderosos.
Para que la asignación de los trabajos sea óptima, es necesario conocer las características de hardware del equipo donde se ejecuta el trabajador, por lo tanto, hay que incluir un módulo de detección de hardware en el programa trabajador para que cada vez que éste solicite una nueva labor, proporcione de forma adicional, el perfil de hardware donde se ejecuta al maestro que le asignará el trabajo adecuado.

Un aspecto que no se ha mencionado hasta el momento es cómo el maestro lleva el registro de los trabajos asignados. En primer lugar, es necesario que todos los trabajos tengan una clave de identificación, numérica o alfanumérica para poder distinguir entre ellos. En segundo lugar hay que definir cual es la categoría de hardware que requiere el trabajo para que sea asignado. Una vez conociendo esto se procede a formar varias colas circulares formadas por los trabajos en cada categoría. Cuando un trabajador solicita una nueva labor, se consulta cual es la categoría de trabajo más alta que se le puede asignar, se busca el último trabajo asignado y se le asigna el siguiente trabajo disponible en la cola circular.
Se maneja una cola circular debido a que por las características del cómputo distribuido de alto caudal de procesamiento, que es el cual está orientado Éxigus, no existen prioridades en la asignación de trabajos y todos los trabajos se consideran iguales, salvo por un perfil mínimo de hardware para su realización. Con la cola circular se garantiza que aunque un trabajador falle con su trabajo, exista la posibilidad de asignar el trabajo a otros trabajadores hasta que no se reciban los resultados de éste y por lo tanto, el trabajo salga de la cola circular respectiva.

3.2.6 Tolerancia a fallos en los resultados

Muchas de las investigaciones, en el contexto del cómputo distribuido, relacionadas con la tolerancia a fallos se han enfocado en fallas en las cuales uno o más elementos de procesamiento o en enlaces de transmisión se detienen de forma temporal o permanente. Menos investigaciones se han realizado para solventar fallas en las cuales los procesadores producen datos incorrectos, ya sea sin intención o de forma maliciosa. Esto se debe en parte a que las investigaciones del cómputo paralelo se han orientado hacia ambientes controlados en donde las fallas más comunes, sino es que las únicas, son de hardware.

Actualmente muchas operaciones de cómputo se realizan en sistemas basados en redes en donde los elementos de procesamiento no sólo están distribuidos físicamente, sino que también pertenecen a diferentes personas. Mientras que mecanismos de paridad y checksum sirven para detectar errores de hardware, no son efectivos en contra de ataques intencionales realizados por elementos de procesamiento maliciosos, o saboteadores, los cuales pueden desensamblar el código que produce los checksums y hacer que datos erróneos tengan checksums correctos.

Existen proyectos que no requieren una exactitud del 100% y por lo tanto, pueden producir buenos resultados generales para un porcentaje bajo de error. Estos proyectos incluyen aplicaciones de representación de imágenes en las cuales unas fallas en pixeles pueden resultar imperceptibles, o también problemas relacionados con algoritmos genéticos, los cuales tienden a corregirse por si mismos.

Otra clase de proyectos son aquellos en los que es posible revisar la validez para un resultado dado en un trabajo, en un tiempo mucho menor que el tomado para generar el resultado. De esta forma, aunque puede tomar un poco de tiempo extra obtener un resultado final, se puede garantizar su validez. Ejemplos de esta clase de proyectos involucran problemas de búsqueda, por ejemplo, patrones de inteligencia en señales de radio o la clave que descifra un mensaje. Así cuando un resultado indica que el elemento buscado ha sido encontrado, se puede revisar rápidamente para comprobarlo y determinar si es un resultado correcto o un falso positivo. Desafortunadamente, aunque los falsos positivos no son problema en tales proyectos, los falsos negativos si lo son. Si un trabajador indica que no hay resultados notables para un trabajo, la única forma de corroborarlo es volver a realizar

el trabajo. En los proyectos donde sólo se espera un resultado único para una búsqueda, este problema se soluciona volviendo a asignar todos los trabajos hasta obtener el resultado correcto.

Los ataques maliciosos pueden venir de un saboteador interno o un falsificador externo. Un falsificador externo es un nodo no registrado como trabajador válido, pero que sin embargo, envía mensajes y resultados que parecen como si vinieran de un trabajador válido. La falsificación puede prevenirse usando firmas digitales, las cuales permiten determinar de forma correcta el origen de la información enviada por un trabajador. Por otro lado, las firmas digitales también permiten asegurar que los datos que viajan de un maestro a un trabajador sean en efecto, originados por el maestro.
Las firmas digitales son muy útiles en proyectos distribuidos cerrados, aquellos donde no se permite una participación de cualquiera, para protegerlos de ataques externos. Sin embargo, son ineficaces en proyectos distribuidos abiertos, donde cualquiera, incluso saboteadores, se les permite colaborar. Esto se debe a que las firmas digitales sólo autentican el origen de los datos, no su contenido.

Una forma de autenticar el contenido es por medio de un cálculo de checksum adicional que se debe realizar con cada trabajo. Si el trabajo no se realiza correctamente, el checksum tampoco lo será por lo que se puede generar una alerta. Sin embargo, por más ofuscado que sea el código para generar este checksum, saboteadores experimentados pueden descubrir el algoritmo por medio de técnicas de ingeniería inversa y hacer que sus resultados maliciosos parezcan siempre correctos.

En los casos en los que no es posible confiar en autenticación, cifrado y ofuscación existen dos formas efectivas de tratar con saboteadores inteligentes: la redundancia y la aleatoriedad. La redundancia permite revisar los resultados de un trabajador contra los de otro que realizó el mismo trabajo. Si se asume que hay más trabajadores buenos que saboteadores, es posible identificar los resultados incorrectos y reducir las oportunidades de aceptarlos como buenos. La aleatoriedad neutraliza los planes de saboteadores que se basan en un comportamiento predecible de la entidad maestro.

Hay varias técnicas que utilizan la redundancia y la aleatoriedad, como el voto mayoritario, inspección al azar y credibilidad.

En el *voto mayoritario* un trabajo se retira de la cola circular de asignaciones hasta que se hayan recolectado $2m-1$ resultados para ese trabajo. Se procede entonces a recoger el resultado que tenga mínimo m instancias idénticas de entre las $2m - 1$ copias. Una versión más eficiente consiste en el esquema *primeros-m*, en donde un trabajo se considera resuelto cuando se obtienen m resultados iguales. La probabilidad de tener errores (trabajos con resultados incorrectos) se reduce exponencialmente al incrementar m, siempre y cuando la población de saboteadores sea baja. Para que este esquema pueda funcionar se necesita que m sea mínimo de 2, lo cual implica que el tiempo esperado de término para los cálculos de un proyecto se incrementa mínimo al doble.

En la técnica de *inspección al azar*, el maestro de forma aleatoria asigna labores a trabajadores cuyos resultados conoce, los cuales se conocen como trabajos inspectores. De esta forma, si un trabajador es atrapado proporcionando un mal resultado para un trabajo inspector, el maestro procede a invalidar todos los resultados previos que haya recibido, e incluso, puede poner a ese trabajador en una lista negra, en la cual los trabajadores que se encuentren registrados en ella les son rechazados sus resultados subsecuentes por un periodo de tiempo determinado por el maestro. Con esta técnica la redundancia disminuye notablemente, ya que el número de trabajos inspectores es mucho menor al de los trabajos normales. La probabilidad de error se reduce linealmente con la cantidad de trabajos descubridores que se asignen.

Idealmente un saboteador descubierto debería ser marcado en la lista negra por siempre, sin embargo en la práctica esto no es posible. Existe el problema de la identificación de los trabajadores, la cual puede realizarse por su dirección IP, o dirección de correo electrónico u otros datos especiales. Sin embargo, una identidad puede ser falsificada fácilmente o incluso un saboteador hábil puede robar los datos que conforman la identidad de un trabajador honesto. Si el proyecto de cómputo distribuido tiene un carácter abierto de libre participación, es mucho más sencillo que un saboteador cree constantemente nuevas identidades para conseguir sus objetivos maliciosos. Debido a estos motivos, lo que se procura es mantener la lista negra de trabajadores durante un tiempo limitado para que los trabajadores honestos a los que se les robó su identidad puedan seguir participando; ahora, si el tiempo necesario para crear una nueva identidad es alto, se logra desanimar a los saboteadores obligándolos a permanecer con una identidad ya creada.

La última técnica para tratar con saboteadores hábiles se conoce como *credibilidad*. Básicamente es una combinación inteligente de votación mayoritaria con inspección al azar. La idea básica de la credibilidad se basa en el principio del umbral, el cual dice que "*si sólo se acepta un resultado para un trabajo cuando la probabilidad condicional de obtener resultados correctos sea al menos un umbral 9, entonces la probabilidad de que todos los resultados de un proyecto sean correctos, es al menos 9*".

Para implementar esta técnica se le asignan valores a diferentes objetos en el sistema. La credibilidad de un objeto X, escrito como Cr(X), se define como un estimado de la probabilidad condicional, dado una estado actual de observación del sistema, de que el objeto X es, o pueda dar, buenos resultados. Existen cuatro tipos diferentes de credibilidad: trabajadores (Cr_P), resultados (Cr_R), grupos de resultados (Cr_G) y trabajos (Cr_W). La credibilidad de un trabajador depende de su comportamiento observado, como por ejemplo, el número de trabajos inspectores pasados. La credibilidad de un trabajador determina la credibilidad de sus resultados, los cuales sirven para determinar la credibilidad de los grupos de resultados. La credibilidad de un grupo de resultados (el cual está compuesto por resultados iguales para un trabajo), se calcula como la probabilidad condicional que los resultados sean correctos, dadas las credibilidades individuales de estos resultados y aquellas de los resultados en los otros grupos. Finalmente, la credibilidad del mejor grupo de resultados para un trabajo proporciona la credibilidad del trabajo y proporcionan un estimado de la probabilidad de error que se obtendría si se escogiera ese resultado.

En el transcurso de un proyecto, las credibilidades de los objetos cambian conforme 1) los trabajadores pasan por los trabajos inspectores, 2) se reciben resultados iguales para un mismo trabajo y, 3) los trabajadores son atrapados por ser saboteadores. Eventualmente, asumiendo suficientes trabajadores honestos, la credibilidad para un trabajo se incrementa y cuando se llega al umbral mínimo, es posible tomar un resultado para ese trabajo.

Las fórmulas exactas para calcular la credibilidad de cada objeto son las siguientes:

- **Credibilidad del trabajador**

1) $Cr_p(P) = 1 - \frac{f}{1-f} \cdot \frac{1}{ke}$ Caso cuando el trabajador ha pasado al menos un trabajo inspector

2) $Cr_p(P) = 1 - f$ Caso cuando al trabajador todavía no se le envía un trabajo inspector

En donde k representa el número de trabajos inspectores correctamente pasados por un trabajador P, f representa la fracción de la población que es saboteadora y e es la base de los logaritmos naturales.

- **Credibilidad de los resultados**

3) $Cr_R(R) = Cr_P(R.resolvedor)$

Esta fórmula quiere decir que la credibilidad de un resultado R es igual a la credibilidad del trabajador P que lo resolvió.

- **Credibilidad de un grupo de resultados**

4) $Cr_G(G_a) = Cr_R(R_1)$ Caso cuando sólo hay un resultado

5) $$Cr_G(G_a) = \frac{P(G_a bueno)\prod_{i\neq a} P(G_i malo)}{\prod_{j=1}^{g} P(G_j malo) + \sum_{j=1}^{g} P(G_j bueno)\prod_{i\neq j} P(G_i malo)}$$

Caso en el que hay g grupos de resultados

La fórmula 5 se usa cuando los resultados obtenidos son diferentes y forman g grupos. $P(G_a bueno)$ es la probabilidad de que todos los resultados en el grupo G_a sean correctos y $P(G_a malo)$ es la probabilidad de que todos los resultados sean incorrectos. Las fórmulas para esos cálculos son:

6) $P(G_a bueno) = \prod_{i=1}^{ma} Cr_R(R_{ai})$

7) $P(G_a malo) = \prod_{i=1}^{ma} (1 - Cr_R(R_{ai}))$

En donde ma es el número de resultados que contiene el grupo G_a y R_{ai} es el resultado i del grupo G_a.

- **Credibilidad de un trabajo**

8) $Cr_W(W) = \max(Cr_G(G_a), Cr_G(G_b), \ldots, Cr_G(G_Z))$

Es decir, la credibilidad para un trabajo W está dada por la credibilidad más alta de todos los grupos de resultados.

La siguiente figura ilustra mejor los conceptos relacionados con credibilidad:

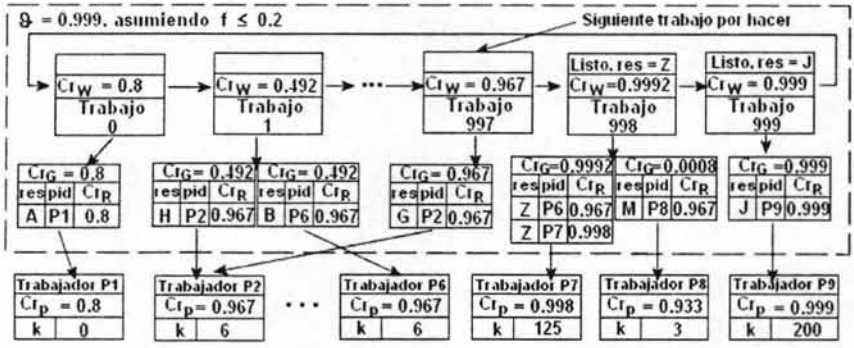

figura 3.5 Ejemplo de cola de asignación de trabajos por credibilidad

En la parte superior se indica que la población considerada como saboteadora, *f*, es menor al 20% (0.2). La letra *k* representa el número de trabajos inspectores que ha pasado correctamente un trabajador. Se observan las distintas credibilidades para varios trabajos. Por ejemplo, el trabajo 0 sólo ha sido resuelto por un trabajador con una credibilidad de 0.8, pero debido a que el umbral para aceptar un trabajo como correcto es de 0.999, todavía no puede ser marcado como resuelto. En el caso del trabajo 1, se han obtenido dos resultados distintos. Como ambos trabajadores a los que se les asignó el trabajo han pasado el mismo número de trabajos descubridores, y por tanto tienen la misma credibilidad, no se puede determinar cual de los resultados es correcto, por lo cual habrá que esperar por más resultados. El caso del trabajo 998 es más interesante. Aquí se muestra que se han obtenido 3 resultados, 2 de los cuales son iguales y uno diferente. Basándose en la credibilidad de los resultados se determinó la credibilidad para cada grupo de resultados. Ya que el grupo con el resultado Z es el que tiene la credibilidad más alta y un umbral suficiente, se marca el trabajo como resuelto y se determina que el trabajador P8 es un saboteador. Por último, como el trabajador P9 tiene una credibilidad de 0.999, prácticamente es un hecho que todos los trabajos que se le asignen tienen el mismo grado de certidumbre de que sean correctos.

3.2.7 Protección del trabajador y los datos en tránsito

Para que un trabajador pueda realizar un trabajo, es necesario que ejecute código del cual no se tiene certeza que es lo que pueda realizar. En el mejor de los casos ese código terminará su ejecución sin ningún contratiempo, sin embargo, es posible que ese código ejecutable también ocasione daños en el equipo donde se ejecuta o incluso ocasionar daños a otros equipos.

Los riesgos que se pueden presentar por ejecutar código pueden ser:

- Consumo excesivo de recursos en el equipo anfitrión ocasionando una baja respuesta del equipo.
- Colapsos en el equipo anfitrión al usar instrucciones incorrectas
- Daño o borrado de archivos del sistema o de usuario

Los tres riesgos anteriores pueden presentarse con o sin intención por parte del programador que realizó el programa, sin embargo, la siguiente lista muestra aquellas acciones en las que claramente el objetivo es ocasionar un daño:

- Robo de información confidencial del usuario donde se ejecuta el trabajador.
- Secuestrar el equipo anfitrión para utilizarlo como miembro en un ataque de negación de servicio contra otro equipo.
- Uso del equipo para distribución de virus
- Uso del equipo para almacenaje de software o contenido ilegal y penalizado
- etc.

Constantemente se descubren nuevos riesgos de seguridad, por lo que es imposible que un software esté diseñado para cubrirlos todos. Lo que se puede hacer es minimizar estos riesgos a niveles aceptables. Algunas de las medidas más comunes son las siguientes:

- Ejecutar cualquier código en una "caja de arena" en la cual se aísla el equipo donde se ejecuta.
- Probar de forma adecuada el código desarrollado para evitar errores de programación
- Autenticar el origen de los datos que se reciben, sea código ejecutable o cualquier otra clase.
- Utilizar un equipo exclusivo en el que no exista información importante.
- Hacer que el código se ejecute con el menor número de privilegios posibles.

- Monitorear constantemente los archivos recibidos por medio de una comunicación en red
- Hacer consiente al dueño de un equipo de los riesgos que pueden existir

No siempre es posible seguir o aplicar las medidas anteriores. Uno de los ambientes de ejecución de "caja de arena" más pequeños es la máquina virtual de Java (JVM, por sus siglas en inglés). JVM en su versión 1.2 ocupa alrededor de 4 MB y generalmente no viene instalado con el sistema operativo, lo cual representa aproximadamente 12 minutos de espera en una descarga con una conexión a Internet de 56 kbps. Esto puede representar una clara desventaja en proyectos de carácter abierto, en donde para maximizar la participación hay que minimizar la cantidad de información que hay que transmitir y descargar. Además de que el código que ejecuta son binarios especiales realizados en lenguaje Java por lo que en algunos casos habría que rescribir gran cantidad de código.

Destinar un equipo exclusivo para que haga las funciones de trabajador tampoco es una medida muy realista, así como tampoco monitorear constantemente los archivos transmitidos y recibidos.

Una de las mejores medidas es autenticar el origen de los datos recibidos en una transferencia de información. Es relativamente fácil de implementar usando firmas digitales y no involucra una sobrecarga notable en el desempeño general, tanto del trabajador como del maestro. Pero indudablemente la mejor medida es concientizar al dueño de un equipo de los riesgos a los cuales se puede enfrentar, para que así evalúe y decida lo que le resulte más conveniente.

En algunos proyectos de cómputo distribuido se cifran todos los datos que viajan del maestro al trabajador y viceversa. El cifrado de los datos los protege contra espionaje y posibles alteraciones. Pero mientras no exista control sobre los trabajadores, estos tranquilamente pueden recibir los datos que legítimamente pueden descifrar, y compartirlos, una vez descifrados, con quien les plazca. Además de que la sobrecarga de operaciones por el cifrado y descifrado puede resultar significativa en el desempeño general del proyecto.

Un argumento muy interesante para evitar cifrar los trabajos se basa en lo siguiente: en el cómputo de alto caudal de procesamiento, que es al cual está orientado la infraestructura

Éxigus, se debe realizar una gran cantidad de trabajo computacional para obtener resultados que ocupan relativamente poco espacio de almacenamiento. Poniendo un ejemplo, en el proyecto SETI@home, todos los días se generan alrededor de 2 Gigabytes de información, de los radiotelescopios, que hay que analizar. Aunque alguien decidiera "robar" estos datos por medio de espionaje en las transmisiones de red, necesitaría mínimo la misma capacidad de cómputo de SETI@home para afectar los intereses del proyecto.
La clase de datos que en los que hay mas interés en proteger contra espionaje son los resultados. Tal vez sea conveniente que se cifren para evitar espionajes externos. Una medida de ese tipo debe ser valorada por los desarrolladores de un proyecto de cómputo distribuido.

3.2.8 Protección del maestro contra ataques de negación de servicio

Un ataque de negación de servicio se realiza contra un sistema o una red y ocurre cuando uno o varios usuarios maliciosos consumen demasiados recursos de tal forma que no quedan suficientes para los demás. Los recursos atacados pueden incluir tiempo de procesador, espacio en disco duro, memoria, ancho de banda en red, etc. Él o los atacantes usan herramientas automatizadas para intentar acceder a los recursos de red expuestos, como un servidor Web.

En el caso de una infraestructura para proyectos de cómputo distribuido, las siguientes acciones ocasionadas repetitivamente pueden originar ataques de negación de servicio:

- Entregar resultados de los cuales no existe una asignación valida de trabajo.
- Solicitar trabajos de forma continua sin haber realizado los antes recibidos.
- Solicitar trabajos y entregar resultados de forma inmediata.
- Entregar resultados de tamaño desmesurado para un trabajo legalmente asignado.

La primera línea de defensa contra estos ataques consiste en proporcionar a todos los trabajadores que forman parte del proyecto, una clave de identificación. Para que un trabajador pueda solicitar trabajo y enviar resultados es necesario que se identifique primero con su clave. Esta no es una medida infalible, ya que esta clave puede ser robada, sin embargo minimiza en gran parte ataques genéricos contra apertura de puertos de comunicación en el maestro.

La siguiente medida de defensa consiste en que a un trabajador al que recién se le asignó su clave de identificación, necesita cumplir con un tiempo mínimo para que se le pueda asignar un trabajo. Este tiempo puede ser del orden de algunos minutos y desanima a aquellos atacantes que piensan causar destrozos basándose en un gran número de identidades diferentes.

Otra medida muy simple está relacionada con evitar recibir resultados para trabajos los cuales no fueron asignados al trabajador. De esta forma, si un trabajador quiere enviar un resultado incorrecto, debe pasar primero por la etapa de solicitud y asignación, la cual consume cierto tiempo.

Debe existir un tiempo mínimo, del orden de segundos o minutos, para la asignación de una labor a un trabajador, después de haber sido asignada una labor anterior o recibir un resultado previo. No es recomendable evitar asignar labores a aquel trabajador que ya se le asignó un trabajo y no ha regresado los resultados correspondientes. Esto es debido a que al propietario del equipo se le puede ocurrir alterar su sistema de alguna forma (por ejemplo, desinstalar el programa trabajador y volverlo a instalar) y por lo tanto le sea imposible completar el último trabajo asignado.

También debe existir un tiempo mínimo para la recepción de resultados, contado a partir del momento en que se realiza la asignación de un trabajo. Con que este tiempo sea de un par de minutos se evitan saturaciones por asignación y recepción de resultados.

Por último debe establecerse un límite máximo de tamaño para los resultados que se recibirán. El administrador de un proyecto debe saber cual es tamaño esperado de los resultados que se obtengan, y por lo tanto, fijar un límite basándose en ese conocimiento.

3.2.9 Balanceo de la carga

En los proyectos de carácter abierto con libre participación existe el problema de que a mayor participación, más difícil es proporcionar los servicios adecuados y eficientes a los trabajadores que colaboran. Existe un número finito de solicitudes, tanto de asignación como recepción que se pueden realizar por segundo. Este número depende en gran medida

del software que se encarga de las solicitudes, así como del hardware donde se ejecuta este software. Cuando el número de solicitudes sobrepasa este límite, simplemente se desechan, lo que a la larga puede representar dejar fuera del proyecto a trabajadores útiles y por lo tanto, incrementar el tiempo que toma concluir un proyecto.

Una forma de incrementar estos límites es por medio de una optimización del software. Utilizando algoritmos óptimos se pueden minimizar los tiempos que se invierten por cada trabajador. Sin embargo, la optimización puede ser difícil de lograr o incluso aunque se logrará, no mejoraría en gran medida el desempeño general.
Las optimizaciones de software no son lo más común. Si se cuentan con recursos monetarios suficientes, casi siempre se prefiere mejorar la plataforma de hardware usada. Estas mejoras contemplan incrementar la velocidad del procesador o su número, aumentar la memoria RAM, usar arreglos de discos redundantes e incluso incrementar el ancho de banda.

Si la infraestructura de hardware con la que se cuenta para un proyecto, así como los recursos monetarios son limitados, la escalabilidad no es una opción. Debe entonces de buscarse alternativas para procurar manejar la mayor cantidad de trabajadores posibles.

Aunque no se pueda incrementar el número de solicitudes por segundo que se puedan atender, existe una alternativa muy simple para incrementar el número de trabajadores que se pueden mantener. Esta alternativa consiste en distribuir las solicitudes a lo largo del tiempo. Para ello es necesario indicarles a los trabajadores que esperen un tiempo mínimo diferente antes de realizar una nueva solicitud, de esa forma se evita una saturación de solicitudes en un instante dado. De forma adicional, si se les proporciona más de una labor a un trabador por cada vez que solicitan trabajo, se decrementa el número de solicitudes relacionadas con asignaciones de trabajo.

Si se cuenta con dos o más equipos físicos en los cuales ejecutar una copia del maestro, es posible atender más trabajadores en general. En tal caso la carga de trabajos se reparte entre los diferentes maestros, así como los trabajadores. Para repartir los trabajadores, es necesario indicarles a los mismos que a partir de un momento predeterminado, sus solicitudes las hagan hacia otro maestro. Esto es relativamente simple de lograr, sólo hay

que indicarles la dirección IP del maestro al cual deben dirigir sus solicitudes en el momento que hagan una solicitud cualquiera a su maestro predeterminado.

Aunque en general el proyecto pudiera verse beneficiado con un incremento en el número de maestros, existe el problema de balancear la carga de trabajos cuando se quiera integrar un nuevo maestro una vez iniciado el proyecto. También se puede dar el problema de que un maestro puede terminar con sus trabajos antes que los demás. Realmente el balanceo dinámico de la carga requiere de conocimientos especializados y estudios basados en la naturaleza de los trabajadores y los trabajos por realizar, por lo que su implementación adecuada representa un reto por sí misma.

3.3 DISEÑO DE LA ARQUITECTURA

Tomando en cuenta las consideraciones anteriormente presentadas, el siguiente diagrama muestra el diseño general de la arquitectura para la infraestructura Éxigus.

figura 3.6 Arquitectura general de la infraestructura Éxigus

Aparte de los trabajos y resultados hay dos nuevos datos que se mueven entre las entidades: los mensajes y las aplicaciones. En el trabajador existen varios módulos los cuales sirven para actuar conforme a los datos que se manejan entre el maestro y el trabajador. Del lado del maestro se observan tres componentes: el servidor Web de archivos, la base de datos y el

Coordinador, este último encargado de manejar a los dos primeros componentes, así como tomar decisiones respecto a los trabajadores, los trabajos asignados y los resultados recibidos.

Los detalles de este diseño se dividirán en tres partes: datos en tránsito, entidad trabajador y maestro, los cuales se mostrarán a continuación.

3.3.1 Datos en tránsito

Los datos en tránsito están formados por los trabajos, los resultados, los mensajes y las aplicaciones. Primero se detallarán las aplicaciones y mensajes, para posteriormente tratar los trabajos y los resultados.

Las aplicaciones contienen simples programas ejecutables por sí mismos empacados en archivos con formato Zip, los cuales se encargan de realizar los trabajos. En este archivo Zip se incluyen todos los archivos necesarios para que el o los ejecutables que contenga puedan correr adecuadamente, como pueden ser archivos de configuración o inicialización, bibliotecas de enlace dinámico, etc. Junto con estos archivos se encuentra uno adicional denominado *signature*, el cual es la firma digital de los archivos contenidos, excepto por él mismo. Aunque no existe obligación alguna en cuanto al nombre que recibe el archivo comprimido con la aplicación, se sugiere usar nombres que indiquen la versión de la aplicación, como por ejemplo *factorial-1.0.0.zip* y *render-3.23.45.zip*.

Los mensajes son los medios por los cuales se puede intercambiar información de poco tamaño entre el trabajador y el maestro. Los mensajes son simplemente secuencias de bytes estructuradas en formato XML. En la infraestructura Éxigus están definidos 8 mensajes:

1. *idWorkerRequest*
2. *idWorkerAsign*
3. *jobRequest*
4. *jobAsign*
5. *workerUpdate*
6. *waitSomeTime*
7. *changeMaster*

8. failReport

1. El mensaje *idWorkerRequest* es enviado por el trabajador hacia el maestro cuando un el trabajador se integra por primera vez al proyecto. Básicamente es la solicitud de una clave de identificación para poder realizar solicitudes posteriores. Su formato es como aparece a continuación:

```
<EXIGMSG type = "idWorkerRequest">
</EXIGMSG>
```

2. El mensaje *idWorkerAsign* es enviado por el maestro al trabajador en respuesta al mensaje *idWorkerRequest*. En este mensaje se incluye la clave de identificación que usará el trabajador que la solicitó. Su formato es el siguiente:

```
<EXIGMSG type="idWorkerAsign" idWorker="clave del trabajador"
        digitalSign = "firma digital del mensaje">
</EXIGMSG>
```

en donde *clave del trabajador* se sustituye en un mensaje real por una secuencia de caracteres alfanumérica determinada por el maestro y el atributo *digitalSign* contiene la representación en formato hexadecimal, en donde dos caracteres expresan un byte de la firma digital del mensaje. Todos los mensajes que son enviados del maestro al trabajador llevan este atributo de firma digital. Un ejemplo con datos reales del mensaje anterior es el siguiente:

```
<EXIGMSG type="idWorkerAsign" idWorker="124-CKEOD-PQSSOT-SD9995-23455-A34FC"
        digitalSign = "892ef34cab789376af90764848ecbaage456750982648590987">
</EXIGMSG>
```

3. El mensaje *jobRequest* es enviado por un trabajador hacia el maestro para indicar que desea solicitar nuevos trabajos por realizar. Su formato es el siguiente:

```
<EXIGMSG type = "jobRequest">
    <worker version="version del trabajador" id="clave del trabajador" />
    <cpu name="nombre del procesador" speedMhz="velocidad en Mhz" />
    <so name="nombre del sistema operativo" version="versión del S.O." />
    <ram sizeMB="tamaño en MB de la memoria RAM" />
    <hd freeMB="espacio libre en disco duro" />
    <applications> listado de aplicaciones que tiene el trabajador,
                   separadas por espacios
    <applications>
</EXIGMSG>
```

Los campos a llenar en el mensaje son bastante claros, pero un ejemplo real ilustra mejor:

```
<EXIGMSG type = "jobRequest">
    <worker version="1.0.125" id="124-CKEOD-PQSSOT-SD9995-23455-A34FC" />
    <cpu name="AMD XP" speedMhz="1667" />
    <so name="Windows 2000" version="5.0.2200" />
    <ram sizeMB="192" />
    <hd freeMB="1032" />
    <applications>render-1.2.zip calcPrim-1.3.45.zip factorial-2.0.zip
    </applications>
</EXIGMSG>
```

4. El mensaje *jobAsign* es enviado por el maestro a un trabajador para indicarle que su solicitud de trabajo ha sido atendida y que se le asignan nuevas labores. El formato del mensaje es el siguiente:

```
<EXIGMSG type = "jobAsign" digitalSign = "firma digital del mensaje">
    <jobDownload fileName="trabajo1">
        http://servidorWebMaestro/trabajo1
    </jobDownload>
    <jobDownload fileName="trabajo2">
        http://servidorWebMaestro/trabajo2
    </jobDownload>
        ...
    <jobDownload fileName="trabajoN">
        http://servidorWebMaestro/trabajoN
    </jobDownload>
</EXIGMSG>
```

Se pueden indicar en el mensaje, cualquier número de trabajos para que el trabajador los descargue de la dirección url indicada entre los elementos *<jobDownload> </jobDownload>*. Un ejemplo de un mensaje donde se asignan tres trabajos se muestra a continuación:

```
<EXIGMSG type = "jobAsign"
     digitalSign = "75eef34cab789376af90764848ecbaage456750982648590987">
    <jobDownload fileName="trabajo21.zip">
        http://Exigus.net/trabajos/trabajo21.zip
    </jobDownload>
    <jobDownload fileName="trabajo54.zip">
        http://217.456.789.123/trabajo54.zip
    </jobDownload>
    <jobDownload fileName="trabajo78.zip">
```

```
            http://Éxigus2.net/trabajo78.zip
      </jobDownload>
</EXIGMSG>
```

5. El mensaje *workerUpdate* es enviado por el maestro a un trabajador para indicarle que antes de que le pueda asignar al menos un trabajo, debe descargar ciertas aplicaciones para que pueda realizar trabajos. El formato del mensaje es el siguiente:

```
<EXIGMSG type = "workerUpdate" digitalSign = "firma digital del mensaje">
       <appUpdate fileName="nombreDeLaAplicacion1">
            http://servidorWebMaestro/nombreDeLaAplicacion1
       </appUpdate>
       <appUpdate fileName="nombreDeLaAplicacion2">
            http://servidorWebMaestro/nombreDeLaAplicacion2
       </appUpdate>
             ...
       <appUpdate fileName="nombreDeLaAplicacionN">
            http://servidorWebMaestro/nombreDeLaAplicacionN
       </appUpdate>
</EXIGMSG>
```

Así como el mensaje anterior, se puede tener un listado de *n* aplicaciones para que las descargue el trabajador. Un ejemplo de este mensaje se muestra a continuación:

```
<EXIGMSG type = "workerUpdate"
         digitalSign = "90efef4cab789376af90764848ecbaage456750982648590987">
      <appUpdate fileName="factorial-1.23.11.zip">
            http://Éxigus.net/aplicaciones/ factorial-1.23.11.zip
      </appUpdate>
      <appUpdate fileName="render-1.00.zip">
            http://256.789.456.123/aplicaciones/ render-1.00.zip
      </appUpdate>
</EXIGMSG>
```

6. El mensaje *waitSomeTime* es enviado por el maestro al trabajador en respuesta a que alguna solicitud no puede ser atendida por un intervalo mínimo de tiempo. El formato del mensaje es el siguiente:

```
      <EXIGMSG type="waitSomeTime" timeMillis="tiempo de espera en milisegundos"
                  digitalSign = "firma digital del mensaje">
      </EXIGMSG>
```

En el siguiente mensaje de muestra se le indica a un trabajador que no realice ninguna solicitud hasta que haya esperado al menos 55,456 milisegundos:

```
<EXIGMSG type="waitSomeTime" timeMillis="55456"
   digitalSign = "90efef4cab789376764e0764848ecbaage456750982648590987">
</EXIGMSG>
```

7. El mensaje *changeMaster* es enviado por el maestro al trabajador en respuesta a cualquier solicitud para indicarle que debe cambiar de maestro, ya que el actual no lo atenderá más. El formato del mensaje es el siguiente:

```
<EXIGMSG type="changeMaster" digitalSign = "firma digital del mensaje">
      <master ip = "dirección Ip del nuevo maestro"
              domain = "dominio del nuevo maestro"
              name = "nombre del nuevo maestro" />
      <pubKey>secuencia hexadecimal con la clave pública del nuevo maestro</pubKey>
</EXIGMSG>
```

Se puede omitir alguno de los atributos *ip* o *domain*, pero no ambos. Un ejemplo de esta clase de mensaje se muestra a continuación:

```
<EXIGMSG type="changeMaster"
         digitalSign = "00efef4c456789376af90764848ecbaage456750982648590987">
      <master ip = "256.789.123.456"
              domain = "masterlobo2.net"
              name = "Lobo2" />
      <pubKey>00efef4c456789376af90764848ecbaage4567509826485909870 0efef4c456789376af9
      0764848ecbaage4567509826485909870 0efef4c456789376af90764848ecbaage45675098264859
      098700efef4c456789376af90764848ecbaage4567509826485909870 0efef4c456789376af90764
      848ecbaage4567509826485909870 0efef4c456789376af90764848ecbaage456750982648590987
      00efef4c456789376af90764848ecbaage456750982648590987
      </pubKey>
</EXIGMSG>
```

8. El mensaje *failReport* es enviado por el trabajador hacia el maestro para indicarle la lista de archivos que no se pudieron descargar del servidor Web. También en este mensaje se incluye el listado de archivos los cuales tienen firmas digitales incorrectas. El formato del mensaje es el siguiente:

```
<EXIGMSG type="failReport" >
      <fileError fileName = "nombreArchivo1"
                 url = "url donde se descargó o debía descargar el archivo"
                 errorType = "tipo de error" />
      <fileError fileName = "nombreArchivo2"
                 url = "url donde se descargó o debía descargar el archivo"
                 errorType = "tipo de error" />
       ...
      <fileError fileName = "nombreArchivoN"
                 url = "url donde se descargó o debía descargar el archivo"
                 errorType = "tipo de error" />
</EXIGMSG>
```

El atributo *errorType* puede tener dos valores: *notFound* o *incorrectSign* para indicar el motivo por el cual se esta reportando a un archivo. Un ejemplo de este mensaje se muestra a continuación:

```
<EXIGMSG type="failReport" >
      <fileError fileName = "factorial-2.03.zip"
                 url = "http://exigmaster.net/apps/factorial-2.03.zip"
                 errorType = "incorrectSign" />
      <fileError fileName = "trabajo-003.zip"
                 url = "http://192.567.345.123/jobs/trabajo-003.zip"
                 errorType = "notFound" />
</EXIGMSG>
```

Los trabajos son archivos empaquetados que contienen los datos necesarios para que las aplicaciones puedan realizar labores útiles. Un trabajo es un archivo en formato Zip que contiene a su vez los siguientes archivos:

- 1 archivo con extensión *.xml* con la especificación del trabajo
- 0 ó más archivos que conforman los datos de entrada para una aplicación
- 1 archivo con la firma digital del contenido del trabajo, denominado *signature*

El archivo con la especificación del trabajo es un documento XML que contiene información necesaria para ejecutar correctamente cada una de las aplicaciones que forman el trabajo, así como instrucciones para tratar de forma adecuada los resultados. El formato de esta clase de archivos es el siguiente:

1.-Cabecera xml

```
<?xml version="1.0" standalone="yes" ?>
```

2.- Etiqueta que inicia la especificación de un trabajo y una clave para distinguir los diferentes trabajos

```
<Job id = "secuencia alfanumérica para distinguir los trabajos">
```

3.- Sección de dependencias. En ella se indican las aplicaciones que se necesitan en el trabajo actual

3. 1 Etiqueta que marca el inicio de la sección

```
<Dependencies>
```

3. 2 Listado de aplicaciones necesarias. Cada aplicación necesaria se indica como sigue:

```
<App>nombreDeLaAplicacion.zip</App>
```

3. 3 Etiqueta que marca el fin de la sección

```
</Dependencies>
```

4.- Sección de tareas. Una tarea contiene las indicaciones de cómo debe ejecutarse una aplicación.

4. 1 Etiqueta que marca el inicio de la sección que agrupa las tareas

```
<Tasks>
```

4. 2 Una o más definiciones de tareas.

4. 2.1 Etiqueta que marca el inicio de una tarea incluyendo atributos adicionales

```
<Task id="secuencia alfanumérica para distinguir las tareas"
    rightExitCode ="número con el valor de término de aplicación esperado"
    retries = "reintentos que se deben realizar en caso de no obtener el
              valor de término de aplicación esperado"
  timeToRetry = "milisegundos de espera entre reintentos">
```

4. 2.2 Definición de como se debe ejecutar un programa ejecutable contenido en la aplicación. Se incluyen los parámetros que se les pasan a modo de línea de comandos.

```
<Process>ejecutable incluyendo su linea de comandos</Process>
```

4. 2.3 Etiqueta que marca el fin de la definición de una tarea

```
<Task>
```

4. 3 Etiqueta que marca el fin de la sección que agrupa las tareas

```
<Tasks>
```

5.- Sección de resultados. Se indica como se deben agrupar los resultados y la dirección url del servidor Web que recibirá el archivo.

5. 1 Etiqueta que marca el inicio de la sección de resultados

```
<Results>
```

5. 2 Subsección de empaquetado de resultados

5.2.1 Etiqueta que marca el inicio de la sección para empacar los resultados. Incluye varios atributos. El nivel de compresión puede ser un valor entre 0 y 9, inclusive ambos, donde 0

quiere decir que no hay compresión y 9 que la compresión es máxima y por tanto, más tardada.

```
<Package fileName = "nombre que recibe el archivo que empaca los
                     resultados en formato Zip"
         zipLevel = "nivel de compresión Zip del archivo empacado"
         upload = "dirección del servidor Web y el nombre del programa que
                   recibe el archivo">
```

5. 2.2 Listado con uno o más archivos que se formaran parte del archivo empacado de resultados.

```
<FileToPack>nombre del archivo que incluido en el paquete</FileToPack>
```

5. 2.3 Etiqueta que marca el fin de la subsección de empaquetado

```
</Package>
```

5. 3 Etiqueta que marca el fin de la sección de resultados

```
</Results>
```

6.- Etiqueta que finaliza la especificación del trabajo

```
</Job>
```

A continuación se muestra una especificación de trabajo con datos reales:

```
<?xml version="1.0" standalone="yes" ?>
<Job id='idJob004'>
  <Dependencies>
    <App>sumaAyB-1.00.zip</App>
    <App>restaAyB-1.00.zip</App>
  </Dependencies>
  <Tasks>
    <Task id='task01' rightExitCode='0' retries='1' timeToRetry='500'>
       <Process>sumaAyB.exe A.txt S.txt</Process>
    </Task>
    <Task id='task02' rightExitCode='0' retries='1' timeToRetry='500'>
       <Process>restaAyB.exe A.txt R.txt</Process>
    </Task>
  </Tasks>
   <Results>
    <Package fileName='job-idJob004.zip' zipLevel='9'
                                upload='http://exigusmaster.net/cgi-bin/resrecv.exe' >
     <FileToPack>S.txt</FileToPack>
     <FileToPack>R.txt</FileToPack>
    </Package>
   </Results>
</Job>
```

Básicamente en la especificación de trabajo anterior el objetivo es realizar una suma y una resta. Para ello se necesitan primero las aplicaciones *sumaAyB.zip* y *restaAyB.zip*, las cuales contienen respectivamente los ejecutables *sumaAyB.exe* y *restaAyB.exe*.
Ambos ejecutables usan dos parámetros especificados en la línea de comandos para ejecutarse correctamente. El primer parámetro es el nombre de un archivo que contiene dos números a operar, mientras que el segundo parámetro de la línea de comandos indica el nombre del archivo en el cual se guardará el resultado de la operación.

Regresando a la especificación del trabajo, primero se distingue la clave de identificación del trabajo en sí (*idJob004*) en la parte superior. Revisando a continuación se observan las dependencias de aplicaciones necesarias para realizar sumas y restas. Llegando a la definición de tareas se pueden distinguir dos tareas identificadas con *task01* y *task02*. En ambas tareas se especifica que *0* es el código de salida correcto para los ejecutables *sumaAyB.exe* y *restaAyB.exe*. También se señala que en caso de error en la ejecución se realice un reintento por tarea, con una espera mínima de 500 milisegundos.
Por la forma en que trabajan los ejecutables se sabe a partir de la línea de comandos definida en el elemento `process` se producirán dos archivos con resultados: *S.txt* y *R.txt* los cuales se originan de *sumaAyB.exe* y *restaAyB.exe* respectivamente. Siguiendo la especificación del trabajo se observa como ambos archivos se empacarán con un nivel de compresión 9 en un archivo denominado `job-idJob004.zip`, el cual se transmitirá vía http hacia un programa especial indicado por la dirección url `http://exigusmaster.net/cgi-bin/resrecv.exe`.

Los resultados, de forma similar a los trabajos son archivos empaquetados que contienen los datos de salida obtenidos al término de la ejecución de un programa. Un resultado es un archivo en formato Zip que contiene a su vez los siguientes archivos:

- 1 archivo con un informe denominado *jobReport.rpt*
- 0 ó más archivos de salida generados al término de la ejecución de un programa, indicados en la especificación de trabajo.

El informe de resultados, *jobReport.rpt*, es un documento XML con el siguiente formato:

1.-Cabecera xml

```
<?xml version="1.0" standalone="yes" ?>
```

2.- Etiqueta que inicia el informe de resultados. Incluye varios atributos.

```
<JobReport id="secuencia alfanumerica que identifica al trabajo"
           idWorker ="clave del trabajador"
           numTasks= "número total de tareas que conforman el trabajo"
           tasksFailed = "número de tareas que no produjeron un código de salida
                          correcto"
           totalTime = "número de segundos invertidos en realizar todas las tareas">
```

3.- Listado de las tareas que conforman el trabajo (se toman del archivo de especificación de trabajo respectivo) indicando el tiempo que tomó terminar la ejecución de la aplicación respectiva y en caso de falla se indica el código de salida final.

```
      <Task id="secuencia alfanumerica que identifica a la tarea"
            fail="indicación si hubo falla o no(yes,no)"
            exitCode = "código de salida final (solo se agrega este atributo en caso
                        de falla) "
            time="segundos que tardó la ejecución en terminar"/>
```

3.- Etiqueta que termina el informe de resultados.

```
</JobReport>
```

Ejemplo:

```
<?xml version="1.0" standalone="yes" ?>
<JobReport id="idJob003"
           idWorker = "124-CKEOD-PQSSOT-SD9995-23455-A34FC "
           numTasks="3" tasksFailed="1" totalTime="75">
    <Task id="idTask01" fail="no" time="29"/>
    <Task id="idTask02" fail="yes" exitCode = "3" time="12"/>
    <Task id="idTask03" fail="no" time="34"/>
</JobReport>
```

3.3.2 Diseño del trabajador

Para el diseño del trabajador primero se empezó por definir el diagrama de flujo de datos o DFD con un grado de detalle de hasta nivel 2. Con esa información fue posible determinar cuales eran los componentes que lo conformaban. Antes de mostrar los DFD correspondientes hay que señalar que la simbología usada es la siguiente:

Símbolo	Descripción
1.4 Proceso	**Proceso.** El proceso muestra una parte del sistema que transforma entradas en salidas; es decir, muestra cómo es que una o más entradas se transforman en salidas. Se incluye un número para identificarlo.
trabajo	**Flujo.** El flujo se usa para describir el movimiento de bloques o paquetes de información de una parte del sistema a otra.
resultados	**Almacén.** El almacén se utiliza para modelar un conjunto de paquetes de datos en reposo.
Maestro	**Terminador.** Los terminadores representan entidades externas con las cuales el sistema se comunica.
Maestro	**Terminador repetido.** Con fines de claridad en los diagramas se usa el símbolo de la izquierda para indicar un mismo terminador ya presentado antes.
trabajos	**Almacén repetido.** Similar al terminador repetido. Indica el mismo almacén presentado de forma anterior.

Ahora sí, con la simbología adecuada ya presentada, se muestran a continuación los varios DFD que conforman el diseño de datos del trabajador.

DFD nivel 0

figura 3.7 DFD de nivel 0 ó de contexto para el Trabajador

Se puede observar el comportamiento general del trabajador: le envían trabajos de parte del maestro y salen resultados hacia el mismo maestro. Aquí los mensajes son el medio por el cual se lleva a cabo la comunicación y coordinación entre trabajador y maestro.

DFD nivel 1

En este diagrama se muestra un mayor nivel de detalle de los procesos, flujos y almacenes de datos que conforman el Trabajador. Salvo por el proceso 3, *Enviar resultados*, el cual se considera primitivo, los otros dos procesos se detallarán aún más de forma separada en un DFD de nivel 2 respectivo. La documentación de los elementos que intervienen en este DFD

de nivel 1 se muestra a continuación. Solamente hace falta aclarar que cuando no se indica el flujo de datos que entra o sale de un almacén, quiere decir que en ese flujo viajan todos los datos que contiene el almacén.

figura 3.8 DFD de nivel 1 del Trabajador

Documentación del diagrama de flujo de datos (DFD)

Nivel: 1
Diagrama proceso: General (todo el Trabajador).
Terminadores: Maestro.
Flujos de datos: Explícitos: mensajes, trabajo, aplicación, resultado.
Implícitos: clave trabajador, plataforma ejecución, datos maestro.

Almacenes de datos: clave trabajador, plataforma ejecución, datos maestro, trabajos, aplicaciones, resultados.
Procesos: Obtener trabajos, realizar trabajos, enviar resultados.

Terminadores

Nombre: Maestro.
Descripción: Su función es la de asignar trabajos, recibir resultados y llevar control de los anteriores. Se comunica con los trabajadores por medio de mensajes.
Flujos que genera: mensajes, trabajos.
Flujos que recibe: mensajes, resultados.

Flujos de datos

Nombre: mensajes.
Descripción: Secuencias de bytes de poco tamaño con las cuales se mantiene una comunicación entre el trabajador y el maestro. Para más información consultar la sección *datos en tránsito*, en este documento.
Fuente: proceso 1(obtener trabajos), terminador Maestro.
Destino: Terminador Maestro, proceso 1(obtener trabajos)

Nombre: trabajo
Descripción: Instrucciones que indican lo que debe hacer un trabajador. Para más información consultar la sección *datos en tránsito*, en este documento.
Fuente: Terminador Maestro, almacén trabajos.
Destino: proceso 1(obtener trabajos) y 2(realizar trabajos), almacén trabajos.

Nombre: aplicación
Descripción: Parte de los medios que le permiten al trabajador realizar un trabajo. Para más

información consultar la sección *datos en transito*, en este documento.
Fuente: Terminador Maestro, almacén aplicaciones.
Destino: proceso 1(obtener trabajos) y 2(realizar trabajos), almacén aplicaciones.

Nombre: resultado
Descripción: Producto de la realización de un trabajo. Para más información consultar la sección *datos en transito*, en este documento.
Fuente: proceso 2 (realizar trabajos), almacén resultados.
Destino: proceso 3 (enviar resultados), almacén resultados y terminador Maestro.

Almacenes de datos

Nombre: clave de trabajador
Descripción: Solamente contiene la clave que proporciona el Maestro para indicar que un trabajador se ha dado de alta en un proyecto. Esta clave se utiliza en todos los mensajes que envía el trabajador para indicar el origen de los mismos.
Nombre: datos maestro
Descripción: Contiene la información necesaria para contactar y autenticar al maestro.
Contenido: nombre del maestro, dirección IP, dominio y clave pública.
Nombre: plataforma ejecución
Descripción: Contiene las características de hardware donde se ejecuta el trabajador, así como software en particular.
Contenido: versión del trabajador, sistema operativo anfitrión, modelo del procesador, velocidad del procesador, cantidad de memoria RAM, espacio libre en disco duro, listado de aplicaciones (aquellas que son necesarias para un proyecto de cómputo distribuido).
Nombre: aplicaciones
Descripción: Contiene una o varias aplicaciones necesarias para el proyecto actual en que colabora el trabajador.
Contenido: Uno o varios registros que incluyen el nombre de la aplicación y archivo Zip que la contiene.
Nombre: trabajos
Descripción: Contiene los trabajos por realizar e información adicional sobre los mismos.
Contenido: Uno o varios registros que incluyen el nombre del trabajo, su estado actual (no iniciado, iniciado o terminado), aplicaciones que necesita, el archivo Zip que lo contiene (en caso de no estar iniciado) o el directorio donde está descomprimido (en caso de estar iniciado y preparado para su realización)
Nombre: resultados
Descripción: Contiene una o varios archivos de resultados generados por el trabajador.

Procesos

Nivel: 1
Número: 1
Nombre: Obtener trabajos
Descripción: Se encarga de dar de alta al trabajador ante el maestro y solicitar aplicaciones y trabajos para hacer. También atiende los casos en los que hay que cambiar de maestro o esperar un tiempo determinado antes de realizar una solicitud.
Entradas: clave trabajador, plataforma ejecución, datos maestro.
Salidas: clave trabajador, datos maestro, mensajes, trabajo, aplicación
Mini especificación:

```
1 SI hay clave de trabajador
2   Enviar mensaje de solicitud de trabajo
3   SELECCIONAR mensaje respuesta
4       CASO actualizar aplicaciones
```

```
 5                Descargar aplicaciones
 6          CASO trabajos asignados
 7                Descargar trabajos
 8          CASO esperar
 9                Esperar el tiempo indicado
10                IR a 2
11          CASO cambiar maestro
12                Actualizar datos del nuevo maestro
13                IR a 2
14 SINO hay clave de trabajador
15    Enviar mensaje para dar de alta
16    Recibir y guardar clave de trabajador
17    IR a 2
```

Nivel: 1
Número: 2
Nombre: Realizar trabajos
Descripción: Ejecuta todas las tareas que conforman cada uno de los trabajos en el almacén de datos correspondiente. Al final de la ejecución de todas las tareas de un trabajo, se crea un archivo de resultados y se almacenan para su uso posterior.
Entradas: trabajo, aplicación.
Salidas: resultado.
Mini especificación:

```
1 SI hay un trabajo iniciado
2   Continuar con la realización de las tareas del trabajo
3   Marcar el trabajo como terminado
4   Guardar el resultado en el almacén
5   IR a 1
6 SINO
7    Obtener un trabajo del almacén
8    SI existe un trabajo no marcado como terminado
9       Preparar el ambiente para la correcta ejecución de las tareas
10      Marcar el trabajo como iniciado
11      IR a 1
12 Terminar
```

Nivel: 1
Número: 3
Nombre: Enviar resultados
Descripción: Toma cada resultado del almacén y lo envía al maestro.
Entradas: resultado, clave trabajador, datos maestro
Salidas: resultado
Mini especificación:

```
1 MIENTRAS exista un resultado por enviar
2   Enviar resultado a la dirección del maestro, identificándose antes con la clave de trabajador
3   Eliminar resultado de almacén resultados
4 Terminar
```

DFD nivel 2 para el proceso Obtener trabajos

En la figura 3.9 se muestra el DFD de nivel 2 para el proceso *Obtener trabajos*. Ahí se muestra en mayor detalle los subprocesos y datos que contiene. En la siguiente

documentación de este proceso se omiten los aspectos ya documentados en el DFD de nivel 1.

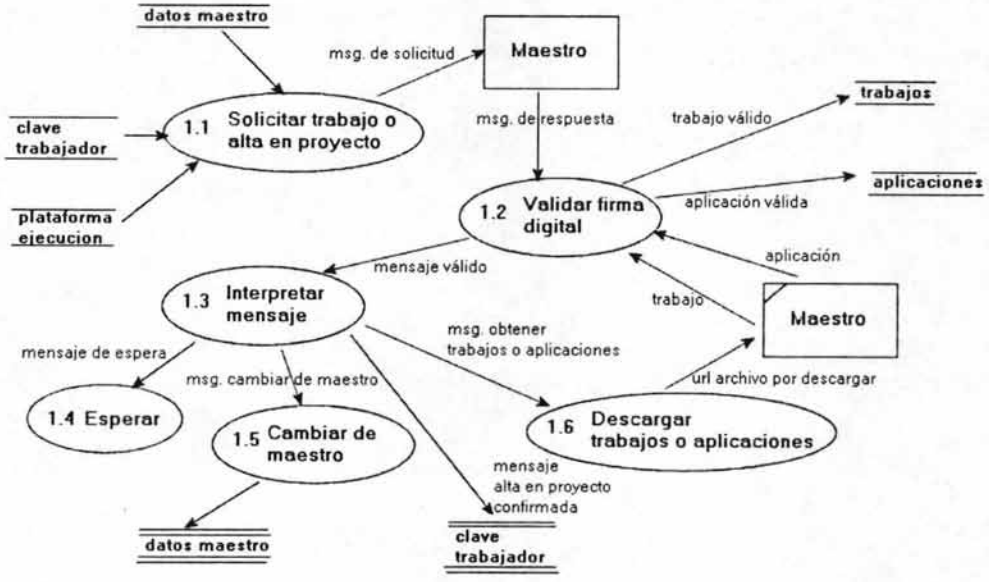

figura 3.9 DFD de nivel 2 del proceso Obtener trabajos

Documentación del diagrama de flujo de datos (DFD)

Nivel: 2
Diagrama proceso: Obtener trabajos.
Terminadores: Maestro.
Flujos de datos: Explícitos: mensaje de solicitud, mensaje de respuesta, mensaje válido, mensaje de espera, mensaje cambiar de maestro, mensaje alta en proyecto confirmada, mensaje obtener trabajos o aplicaciones, url archivo por descargar, trabajo, aplicación, aplicación válida, trabajo válido.
Implícitos: clave trabajador, plataforma ejecución, datos maestro.
Almacenes de datos: clave trabajador, plataforma ejecución, datos maestro, trabajos, aplicaciones.
Procesos: Solicitar trabajo o alta en proyecto, validar firma digital, interpretar mensaje, esperar, cambiar de maestro, descargar trabajos o aplicaciones.

Flujos de datos

Nombre: mensaje de solicitud.
Descripción: Existen dos mensajes de solicitud: uno para solicitar la clave de trabajador (lo que es equivalente a solicitar la alta en un proyecto) y un mensaje para solicitar trabajo. Para más información consultar la sección *datos en transito*, en este documento.
Fuente: proceso 1.1 (Solicitar trabajo o alta en proyecto).
Destino: Terminador Maestro.

Nombre: mensaje de respuesta.
Descripción: Son cuatro mensajes de respuesta a una solicitud:
1)Esperar determinado tiempo
2)Cambiar de maestro
3)Asignación de clave de trabajador (alta en proyecto confirmada)
4)Obtener trabajos o aplicaciones
Para más información consultar la sección *datos en transito*, en este documento.
Fuente: Terminador Maestro.
Destino: Proceso 1.2 (Validar firma digital).

Nombre: mensaje válido
Descripción: Es cualquier mensaje de respuesta del cual se ha autenticado su origen por medio de la firma digital que contiene. Para más información consultar la sección *datos en transito*, en este documento.
Fuente: Proceso 1.2 (Validar firma digital).
Destino: Proceso 1.3 (Interpretar mensaje).

Nombre: mensaje espera
Descripción: Es un tipo de mensaje de respuesta que le indica al trabajador el tiempo mínimo de espera antes de realizar cualquier nueva solicitud. Para más información consultar la sección *datos en transito*, en este documento.
Fuente: Proceso 1.3 (Interpretar mensaje).
Destino: Proceso 1.4 (Esperar).

Nombre: mensaje cambiar de maestro
Descripción: Mensaje que contiene los datos del nuevo maestro que atenderá las solicitudes del trabajador. Para más información consultar la sección *datos en transito*, en este documento.
Fuente: Proceso 1.3 (Interpretar mensaje).
Destino: Proceso 1.4 (Cambiar de maestro).

Nombre: mensaje alta en proyecto confirmada
Descripción: Mensaje que contiene la clave del trabajador que le asigna el maestro para que pueda solicitar trabajos o enviar resultados. Para más información consultar la sección *datos en transito*, en este documento.
Fuente: Proceso 1.3 (Interpretar mensaje).
Destino: Almacén clave trabajador.

Nombre: mensaje obtener trabajos o aplicaciones.
Descripción: Mensaje que le indica la clase de archivo o archivos, así como las direcciones específicas de donde debe descargarlos. Para más información consultar la sección *datos en transito*, en este documento.
Fuente: Proceso 1.3 (Interpretar mensaje).
Destino: Proceso 1.6 (Descargar trabajos o aplicaciones).

Nombre: url archivo por descargar
Descripción: Dirección URL en donde se encuentra un archivo que hay que descargar.
Fuente: Proceso 1.6 (Descargar trabajos o aplicaciones).
Destino: Terminador Maestro.

Nombre: trabajo válido
Descripción: Es cualquier trabajo del cual se ha autenticado su origen por medio de la firma digital que contiene. Para más información consultar la sección *datos en transito*, en este documento.
Fuente: Proceso 1.2 (Validar firma digital).
Destino: Almacén trabajos.

Nombre: aplicación válida
Descripción: Es cualquier aplicación de la cual se ha autenticado su origen por medio de la firma digital que contiene. Para más información consultar la sección *datos en transito*, en este documento.
Fuente: Proceso 1.2 (Validar firma digital).
Destino: Almacén aplicaciones.

Procesos

Nivel: 2
Número: 1.1
Nombre: Solicitar trabajo o alta en proyecto
Descripción: Determina el tipo de mensaje de solicitud que hay que realizar, lo prepara y lo envía.
Entradas: clave trabajador, plataforma ejecución, datos maestro.
Salidas: mensaje de solicitud
Mini especificación:

```
1 SI hay clave de trabajador
2   Enviar mensaje de solicitud de trabajo
3 SINO hay clave de trabajador
4     Enviar mensaje para dar de alta
5 Terminar
```

Nivel: 2
Número: 1.2
Nombre: Validar firma digital
Descripción: Autentica el origen de los datos que se le presentan por medio de la firma digital que contienen.
Entradas: mensaje de respuesta, trabajo, aplicación.
Salidas: mensaje válido, trabajo válido, aplicación válida.
Mini especificación:

```
1 SELECCIONAR tipo de entrada
2   CASO mensaje
3         SI firma digital correcta
4             pasar el mensaje
5         SINO
6             rechazar el mensaje
7   CASO trabajo
8         SI firma digital correcta
9             guardar el trabajo en almacén trabajos
10        SINO
11            rechazar el trabajo
12  CASO aplicación
13        SI firma digital correcta
14            guardar la aplicación en almacén aplicaciones
15        SINO
16            rechazar la aplicación
17 Terminar
```

Nivel: 2
Número: 1.3
Nombre: Interpretar mensaje
Descripción: Llama a un proceso o realiza un acción dependiendo del contenido del mensaje.
Entradas: mensaje válido
Salidas: mensaje de espera, mensaje cambiar de maestro, mensaje alta en proyecto confirmada, mensaje obtener trabajos o aplicaciones.
Mini especificación:

```
1 SELECCIONAR tipo de mensaje
2   CASO mensaje de espera
3         Llamar al proceso 1.4 (Esperar)
4   CASO mensaje cambiar de maestro
5         Llamar al proceso 1.5 (Cambiar de maestro)
6   CASO mensaje alta en proyecto confirmada
7         Guardar la clave de trabajador en el almacén del mismo nombre
8   CASO mensaje obtener trabajos o aplicaciones
9         Llamar al proceso 1.6 (Descargar trabajos o aplicaciones)
```

10 Terminar
Nivel: 2
Número: 1.4
Nombre: Esperar
Descripción: Realiza una espera de un determinado tiempo antes de realizar cualquier otra actividad.
Entradas: mensaje esperar
Salidas: no hay datos de salida
Mini especificación:

1 Extraer el tiempo de espera del mensaje esperar
2 Esperar el tiempo indicado
3 Terminar
Nivel: 2
Número: 1.5
Nombre: Cambiar de maestro
Descripción: Registra los datos del nuevo maestro que atenderá las solicitudes y recepciones.
Entradas: mensaje cambiar de maestro.
Salidas: datos del maestro(se actualiza el almacén)
Mini especificación:

1 Extraer los datos del nuevo maestro del mensaje
2 Actualizar el almacén datos del maestro
3 Terminar
Nivel: 2
Número: 1.6
Nombre: Descargar trabajos o aplicaciones
Descripción: Se encarga de obtener del servidor de archivos correspondiente los archivos indicados por el mensaje.
Entradas: mensaje obtener trabajos o aplicaciones.
Salidas: url archivo por descargar
Mini especificación:

1 Crear un listado de las urls de los archivos por descargar
2 MIENTRAS exista un url de un archivo sin descargar en la lista
3 descargar el archivo del maestro
4 eliminar la url de la lista
5 llamar al proceso Validar firma pasando ese archivo como entrada
6 Terminar

DFD nivel 2 para el proceso Realizar trabajos

En la figura 3.10 se muestra el DFD de nivel 2 para el proceso *Realizar trabajos*. Ahí se muestra en mayor detalle los subprocesos y datos que contiene. La documentación se muestra a continuación.

Documentación del diagrama de flujo de datos (DFD)

Nivel: 2
Diagrama proceso: Realizar trabajos
Terminadores: ninguno
Flujos de datos: nombre de trabajo y estado, trabajo iniciado, trabajo nuevo, resultado, estado del

trabajo, trabajo, aplicación
Almacenes de datos: trabajos, aplicaciones, resultados.
Procesos: Revisar estado trabajos, ejecutar tareas, preparar ambiente trabajo.

figura 3.10 DFD de nivel 2 del proceso Realizar trabajos

Flujos de datos

Nombre: nombre de trabajo y estado
Descripción: Son datos con los cuales se determina el trabajo sobre el cual se está operando.
Fuente: almacén trabajos.
Destino: proceso 2.1 (Revisar estado trabajos)

Nombre: trabajo iniciado
Descripción: Es el nombre de un trabajo que se encuentra listo para que se realicen sus tareas y del que posiblemente ya se hayan terminado algunas tareas que lo conforman.
Fuente: proceso 2.1 (Revisar estado trabajos)
Destino: proceso 2.2 (Ejecutar tareas)

Nombre: trabajo nuevo
Descripción: Es el nombre de un trabajo que no ha sido iniciado y no tiene su ambiente de ejecución preparado.
Fuente: proceso 2.1 (Revisar estado trabajos)
Destino: proceso 2.3 (Preparar ambiente de trabajo)

Nombre: estado del trabajo
Descripción: Indicación de la situación actual de un trabajo(iniciado o nuevo)
Fuente: proceso 2.3 (Preparar ambiente de trabajo)
Destino: almacén trabajos.

Procesos

Nivel: 2
Número: 2.1
Nombre: Revisar estado de trabajo
Descripción: Dependiendo del estado actual de un trabajo, llama al proceso correspondiente, ya sea para continuar su realización o para prepararlo para iniciar su ejecución.
Entradas: nombre de trabajo y estado
Salidas: trabajo iniciado, trabajo nuevo
Mini especificación:

1 SI el estado del trabajo es iniciado
2 Llamar al proceso Ejecutar tareas

3 SINO
4 Llamar al proceso Preparar ambiente de trabajo
5 Terminar

Nivel: 2
Número: 2.2
Nombre: Ejecutar tareas
Descripción: Inicia o continua la ejecución de las tareas que forman un trabajo, al cual se ambiente de trabajo ya ha sido preparado.
Entradas: trabajo iniciado
Salidas: resultado
Mini especificación:

1 MIENTRAS haya tareas sin hacer
2 Realizar tarea
3 Guardar resultado en almacén resultados
4 Terminar

Nivel: 2
Número: 2.3
Nombre: Preparar ambiente de trabajo
Descripción: Prepara en un directorio nuevo todos los archivos necesario (aplicaciones y archivos del trabajo) para que las tareas que conforman un trabajo puedan ejecutarse correctamente.
Entradas: trabajo nuevo, trabajo, aplicación
Salidas: estado del trabajo
Mini especificación:

1 Crear un directorio nuevo
2 Descomprimir los archivos que conforman el trabajo
3 Descomprimir las aplicaciones necesarias al nuevo directorio
4 Actualizar el estado del trabajo en el almacén de trabajos
5 Terminar

3.3.3 Diseño del maestro

Para el diseño del maestro se pensará en él como un todo, a pesar que de antemano se conocen los componentes principales en que se divide. De igual forma que el trabajador, el diseño del maestro comienza con sus diagramas de flujos de datos de nivel 0, 1 y 2, los cuales se muestran a continuación.

En la figura 3.11 se puede observar el comportamiento general del maestro: un productor genera trabajos que se distribuirán posteriormente entre los trabajadores solicitantes. Cuando estos trabajadores terminen sus labores respectivas, el Maestro recibe los resultados correspondientes, los cuales dispondrá el productor como le sea conveniente.

DFD nivel 0

figura 3.11 DFD de nivel 0 ó de contexto para el Maestro

DFD nivel 1

En la figura 3.12 se muestran los dos procesos principales que conforman el Maestro: asignar trabajos y recibir resultados. Ambos procesos se detallarán en DFDs de nivel 2, después de ver la documentación de los elementos del DFD de nivel 1. Como se puede observar, algunos de los elementos que conforman el DFD del Maestro, como los mensajes, trabajos y aplicaciones, ya han sido detallados en el diseño del Trabajador, por lo que serán omitidos en la documentación.

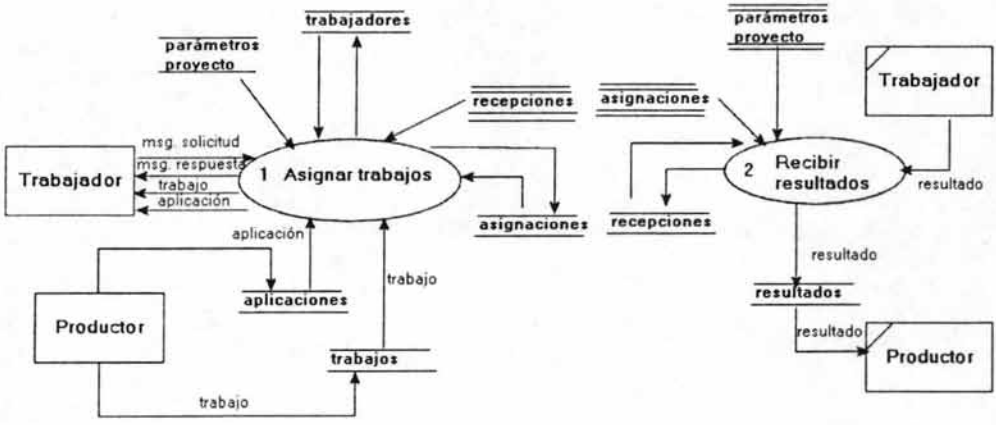

figura 3.12 DFD de nivel 1 del Maestro

Documentación del diagrama de flujo de datos (DFD)

Nivel: 1
Diagrama proceso: General (todo el Maestro).
Terminadores: Productor, Trabajador.

Flujos de datos: Explícitos: mensaje solicitud, mensaje respuesta, trabajo, aplicación, resultado.
Implícitos: asignaciones, recepciones, parámetros proyecto, trabajadores.
Almacenes de datos: parámetros proyecto, trabajadores, aplicaciones, trabajos, resultados, recepciones, asignaciones.
Procesos: Asignar trabajos, Recibir resultados.

Terminadores

Nombre: Productor
Descripción: Se encarga de generar los trabajos que forman el proyecto. También recolecta los resultados que se generen una vez que el Maestro haya determinado que se cumple cierto grado de certidumbre.
Flujos que genera: trabajos.
Flujos que recibe: resultados.
Nombre: Trabajador
Descripción: Solicita trabajos, los realiza y entrega resultados.
Flujos que genera: resultados.
Flujos que recibe: trabajos.

Flujos de datos

Todos los flujos de datos en este nivel han sido documentados en el diseño del trabajador.

Almacenes de datos

Nombre: parámetros proyecto
Descripción: Contiene información que controla la forma en que se asignan, validan y reciben los datos que circulan en el proyecto.
Contenido: método de asignación, método de validación, aplicaciones necesarias, tamaño máximo de resultado, tiempos mínimos de espera, numero máximo de trabajos por asignar.
Nombre: trabajadores
Descripción: Contiene datos relevantes sobre los trabajadores que colaboran en el proyecto.
Contenido: clave del trabajador, indicador de saboteador, plataforma de hardware y software en donde se ejecuta el trabajador.
Nombre: trabajos
Descripción: Contiene los trabajos por asignar, así como la plataforma de hardware necesaria para que se ejecuten. En base a este almacén, se mantiene una cola circular en la cual se lleva la secuencia de trabajos por asignar.
Contenido: clave del trabajo, nombre del trabajo, requerimientos de hardware
Nombre: asignaciones
Descripción: Contiene los registros de los trabajos asignados en el transcurso de vida del proyecto.
Contenido: Nombre del trabajo, clave del trabajador, fecha y hora de asignación del trabajo.
Nombre: recepciones
Descripción: Contiene los registros de los resultado recibidos en el transcurso de vida del proyecto.
Contenido: Nombre del trabajo correspondiente, nombre del resultado, clave del trabajador, fecha y hora de recepción del resultado.

Procesos

Nivel: 1
Número: 1
Nombre: Asignar trabajos.
Descripción: Determina los trabajos que se asignarán a un trabajador que lo solicite. También se encarga de asignar claves de trabajadores.
Entradas: parámetros proyecto, trabajadores, mensaje solicitud, aplicación, trabajo, recepciones, asignaciones.
Salidas: mensaje respuesta, trabajo, aplicación, asignaciones, trabajadores
Mini especificación:

1 SELECCIONAR mensaje solicitud
2 CASO msg. solicitar clave trabajador
3 asignar clave trabajador
4 CASO msg. solicitar trabajo
5 validar requerimientos mínimos
6 determinar trabajos por asignar
7 comunicar trabajos asignados
8 Terminar

Nivel: 1
Número: 2
Nombre: Recibir resultados
Descripción: Registra los resultados que se reciban por parte de los trabajadores los valida.
Entradas: resultado, parámetros proyecto, asignaciones, recepciones,
Salidas: resultado, recepciones
Mini especificación:

1 Recibir resultado
2 Validar requerimientos mínimos
3 Registrar resultado
4 Validar resultados

DFD nivel 2 para el proceso Asignar trabajos

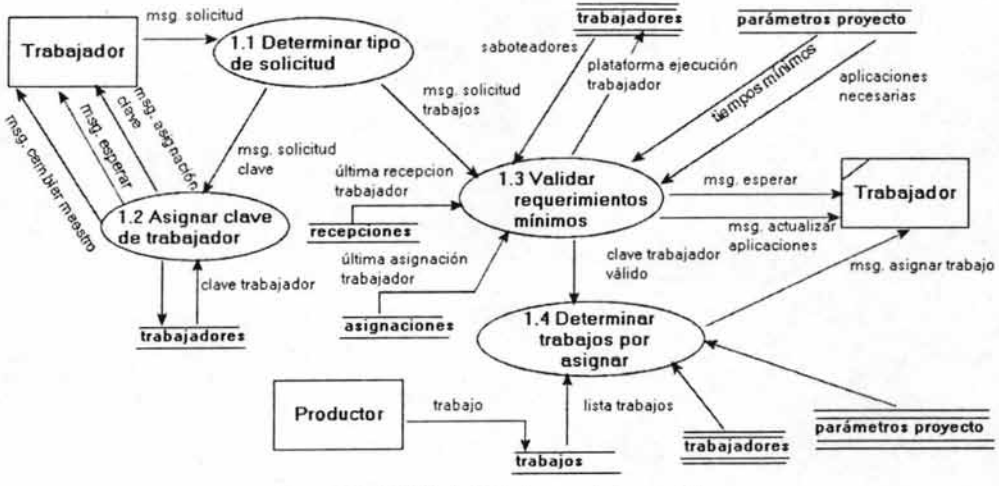

figura 3.13 DFD de nivel 2 del proceso Asignar trabajos

Documentación del diagrama de flujo de datos (DFD)

Nivel: 2
Diagrama proceso: Asignar trabajos.

Terminadores: Trabajador, Productor.
Flujos de datos:
Explícitos: mensaje solicitud, mensaje solicitud clave, mensaje asignación clave, mensaje esperar, mensaje cambiar maestro, clave trabajador, mensaje solicitud trabajos, última recepción trabajador, última asignación trabajador, saboteadores, plataforma ejecución trabajador, tiempos mínimos, aplicaciones necesarias, mensaje actualizar aplicaciones, mensaje asignar trabajo, clave trabajador válido, método de asignación, tiempo deseado de ocupación, lista trabajos, trabajo.
Implícitos: trabajadores, parámetros proyecto
Almacenes de datos: trabajadores, parámetros proyecto, asignaciones, recepciones, trabajos.
Procesos: Determinar tipo de solicitud, Asignar clave de trabajador, Validar requerimientos mínimos, Determinar trabajos por asignar.

Flujos de datos

Nombre: última recepción trabajador
Descripción: Datos del último resultado recibido dado un trabajador.
Fuente: almacén recepciones.
Destino: proceso Validar requerimientos mínimos.

Nombre: última asignación trabajador
Descripción: Datos del último trabajo asignado dado un trabajador.
Fuente: almacén asignaciones.
Destino: proceso Validar requerimientos mínimos.

Nombre: Saboteadores
Descripción: Listado de trabajadores que no entregaron el resultado correcto para un trabajo descubridor.
Fuente: almacén trabajadores.
Destino: proceso Validar requerimientos mínimos.

Procesos

Nivel: 2
Número: 1.1
Nombre: Determinar tipo de solicitud
Descripción: Determina el tipo de mensaje de solicitud que se está recibiendo y llama al proceso adecuado para que lo atienda.
Entradas: mensaje solicitud.
Salidas: mensaje solicitud clave, mensaje solicitud trabajos.
Mini especificación:

1 SELECCIONAR tipo de mensaje de solicitud
2 CASO mensaje solicitud clave
3 Llamar a Asignar clave de trabajador
4 CASO mensaje solicitud trabajos
5 Llamar a Validar requerimientos mínimos
6 Terminar

Nivel: 2
Número: 1.2
Nombre: Asignar clave trabajador
Descripción: Determina el tipo de mensaje de solicitud que se está recibiendo y llama al proceso adecuado para que lo atienda.
Entradas: mensaje solicitud clave, clave trabajador
Salidas: mensaje asignación clave, mensaje esperar, mensaje cambiar maestro.
Mini especificación:

1 SI hay clave de trabajador sin asignar
2 Registrar clave como asignada
3 Enviar mensaje asignación clave

```
4 SINO
5       SI existe un maestro alternativo
6           Enviar mensaje cambiar maestro
7       SINO
8           Enviar mensaje esperar
9 Terminar
```

Nivel: 2
Número: 1.3
Nombre: Validar requerimientos mínimos
Descripción: Determina si se puede asignar o no trabajos a un trabajador dado. También revisa que el trabajador tenga las aplicaciones necesarias para el proyecto.
Entradas: mensaje solicitud trabajos, saboteadores, tiempos mínimos, aplicaciones necesarias, última asignación trabajador, última recepción trabajador.
Salidas: plataforma ejecución trabajador, mensaje esperar, mensaje actualizar, clave trabajador válido.
Mini especificación:

```
1 SI trabajador esta en lista de saboteadores
2     Enviar mensaje esperar
3 SINO
4     SI trabajador cumple con tiempos mínimos
5         Actualizar datos plataforma ejecución trabajador
6         SI trabajador tiene las aplicaciones necesarias
7             Llamar al proceso 1.4 (Determinar trabajos por asignar)
8         SINO
9             Enviar mensaje actualizar
10    SINO
11        Enviar mensaje esperar
12 Terminar
```

Nivel: 2
Número: 1.4
Nombre: Determinar trabajos por asignar
Descripción: Prepara un mensaje indicando el número y los nombres de los trabajos que se asignarán a un trabajador. Para las asignaciones se toma en cuenta básicamente el método de asignación, el tipo de trabajos que se pueden asignar al trabajador y el número máximo de trabajos por asignar.
Entradas: clave trabajador válido, método de asignación, número máximo de trabajos por asignar.
Salidas: mensaje asignar trabajo
Mini especificación:

```
1 Determinar el perfil de hardware del trabajador
2 Determinar el perfil de trabajos adecuados al hardware del trabajador
3 En base al método de asignación, obtener a lo más un número de trabajos igual al número
  máximo de trabajos por asignar
4 Preparar mensaje de asignación de trabajos
5 Enviar mensaje asignar trabajo
6 Terminar
```

DFD nivel 2 para el proceso Recibir resultados

El DFD de nivel 2 para este proceso se muestra en la figura 3.14

Documentación del diagrama de flujo de datos (DFD)

Nivel: 2
Diagrama proceso: Recibir resultados.
Terminadores: Trabajador, Productor.

Flujos de datos:
Explícitos: clave trabajador, datos del resultado, tamaño máximo resultado, tiempos mínimos, última asignación trabajador, última recepción trabajador, trabajador saboteador, datos del resultado verificados, resultado, nuevo registro recepción, método de validación, saboteadores.
Implícitos: resultados no validados, recepciones.
Almacenes de datos: trabajadores, parámetros proyecto, asignaciones, recepciones, resultados no validados, resultados.
Procesos: Validar requerimientos mínimos, Registrar resultado, Validar resultados.

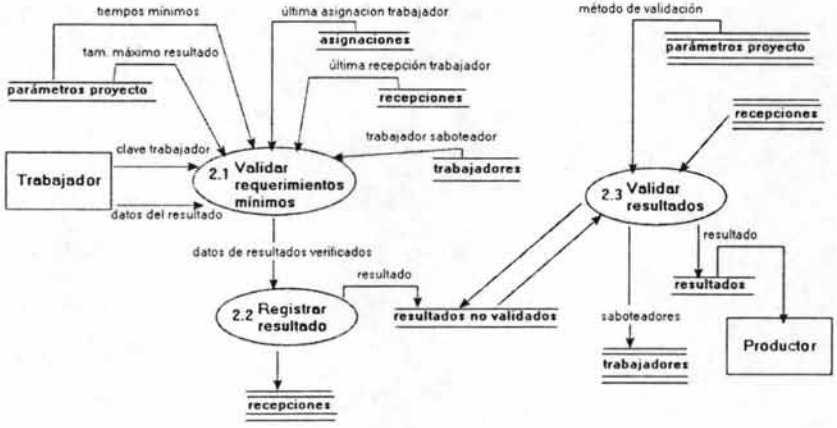

figura 3.14 DFD de nivel 2 del proceso Recibir resultados

Flujos de datos

Nombre: datos del resultado
Descripción: Contiene el reporte de ejecución del trabajo respectivo, los archivos propios de resultados, nombre del trabajo y nombre del resultado.
Fuente: terminador Trabajador.
Destino: proceso 2.1(Validar requerimientos mínimos)

Nombre: tamaño máximo resultado.
Descripción: Límite de espacio que puede ocupar un resultado para que se reciba.
Fuente: almacén parámetros proyecto.
Destino: proceso 2.1 (Validar requerimientos mínimos)

Nombre: trabajador saboteador
Descripción: Indicador si dada una clave de trabajador, el mismo está registrado como saboteador.
Fuente: almacén trabajadores.
Destino: proceso 2.1 (Validar requerimientos mínimos).

Nombre: datos del resultado verificados
Descripción: Son los mismos datos del resultado que pasaron cumplieron con los requerimientos mínimos.
Fuente: proceso 2.1 (Validar requerimientos mínimos).

Destino: proceso 2.2 (Registrar resultado)

Procesos

Nivel: 2
Número: 2.1
Nombre: Validar requerimientos mínimos.
Descripción: Para aceptar un resultado se deben cumplir los siguientes requerimientos:

1)El trabajador debe estar registrado
2)El trabajador no debe ser saboteador
3)El resultado debe corresponder al último trabajo asignado
4)Se deben cumplir con los tiempos mínimos para recibir resultados
5)El resultado no debe ser mayor a un tamaño especificado

Una vez cumplidos con estos requisitos, se puede recibir un resultado para determinar posteriormente su validez para mantenerlo o eliminarlo definitivamente.
Entradas: datos del resultado, clave del trabajador, tamaño máximo resultado, tiempos mínimos, última asignación trabajador, última recepción trabajador, trabajador saboteador.
Salidas: datos de resultado validados.
Mini especificación:

1 SI (el trabajador esta registrado & el trabajador no es saboteador &
2 el resultado corresponde al último trabajo asignado &
3 se cumplen con los tiempos mínimos para recibir resultados &
4 el resultado no es mayor al tamaño máximo para resultados)
5 Llamar al proceso 2.2 Registrar resultado
6 SINO
6 Terminar

Nivel: 2
Número: 2.2
Nombre: Registrar resultado
Descripción: Una vez que se ha determinado el cumplimiento de los resultados, se registra el resultado para su posterior validación.
Entradas: datos de resultado verificados
Salidas: una nueva entrada en almacén recepciones.
Mini especificación:

1 Crear una nueva entrada en almacén recepciones
2 Guardar el resultado en el almacén resultados no validados
3 Terminar

Nivel: 2
Número: 2.3
Nombre: Validar resultados
Descripción: Dependiendo del método de validación, se determinan cuales de los resultados recibidos son válidos. Aquellos que no son válidos son eliminados y los trabajadores que los hicieron pueden o no ser marcados como saboteadores, dependiendo del método de validación.
Entradas: método de validación, recepciones, resultados no validados.
Salidas: resultados no validados, saboteadores, resultado.
Mini especificación:

Dependiendo del método de validación
1 Mover resultados válidos de resultados no validados a resultados
2 Eliminar resultados no válidos
3 Actuar o no contra los trabajadores que entregaron resultados no válidos
4 Terminar

3.4 IMPLEMENTACIÓN DEL TRABAJADOR

El software trabajador se implementó creando varios archivos de código fuente escritos principalmente en lenguaje C, utilizando algunas extensiones útiles de C++ (como definir variables en cualquier parte y no al principio). Debido a que en la biblioteca estándar del lenguaje C, el soporte para la creación y manipulación de procesos es muy limitada, se prefirió utilizar llamadas al sistema operativo, más en concreto, del sistema operativo Microsoft Windows. Esto quiere decir, que el trabajador sólo se ejecuta en la plataforma Windows de 32 bits (posiblemente en la de 64 bits también, aunque no ha sido probado), aunque existe la posibilidad de que el trabajador pueda correr en el sistema operativo GNU/Linux por medio de la aplicación WINE[13] o con el uso de software para una máquina virtual, como el producto VMware[14] Workstation. De cualquier forma, menos del 30% del código que conforma el trabajador depende de llamadas al sistema Windows, así que es viable su modificación para una ejecución nativa en otros sistemas operativos de la familia Unix.

Debido a los diversos requerimientos del trabajador, fue necesario el uso de bibliotecas externas para acelerar y facilitar la implementación. Estas bibliotecas son OpenSSL(para verificación de firmas digitales y transmisión de datos segura), Zlib(para el manejo de archivos en formato Zip), TinyXML (para manipulación de documentos en formato XML) y de Camel (para identificación de fabricante y velocidad de procesador). También se hace uso de la aplicación CuRL para descargar y subir archivos desde y hacia servidores Web.

Toda la funcionalidad del trabajador queda implementada en una biblioteca de enlace dinámico (DLL) que puede integrarse en cualquier programa que pueda hacer uso de una DLL. Esto se hizo con la finalidad de separar la interfaz gráfica de usuario de la lógica de negocios. La biblioteca consta de simplemente 4 funciones: una para iniciar la máquina de estados del trabajador; otra para parar temporalmente la máquina de estados; otra para reanudarla y una última para terminarla totalmente.

---

[13] http://www.winehq.com

[14] http://www.vmware.com

El código fuente esta organizado en 5 directorios:

1. cpuInfo: tiene el código fuente necesario para identificar los procesadores.
2. openssl: contiene sólo los encabezados (archivos .h) para poder utilizar la biblioteca estática que contiene.
3. worker: salvo por las llamadas a otras bibliotecas, este directorio contiene prácticamente todo el código fuente para la implementación de la funcionalidad necesaria del trabajador.
4. xml: aquí se encuentran los fuentes necesarios para manipular documentos XML
5. zip: se incluyen fuentes y bibliotecas para manipulación de archivos zip

No se mencionará aquí la descripción de los archivos fuentes que no se encuentren en el directorio *worker*, ya que existe documentación suficiente. Aunque el código fuente del directorio *worker* está documentado, a continuación se presentará una guía de la forma y el orden en que se llaman las funciones más relevantes del código. Antes de ver esto es necesario hacer notar algo muy importante respecto a archivos y directorios necesarios para la ejecución del trabajador.

3.4.1 Archivos y estructura de directorios necesaria

El trabajador (por simplicidad se referirá a él como si fuera un programa ejecutable por sí mismo y no una biblioteca de enlace dinámico) requiere que exista un directorio especial, que internamente lo llama como *$rutaBase* (esta abreviación tipo Unix quiere decir que la variable en un contexto determinado, se sustituye por su valor, por ejemplo, si *$rutaBase* = *c:\trabajador*, entonces la expresión abreviada *$rutaBase\documentacion*, se traduce como *c:\trabajador\documentacion*). El directorio *$rutaBase* puede estar localizado en cualquier parte, y debe contener a su vez los subdirectorios *$rutaBase\apps*, *$rutaBase\jobs*, *$rutaBase\results*, *$rutaBase\sys*.

También se espera que existan los siguientes archivos:

- $rutaBase\exigGeneral.xml
- $rutaBase\sys\curl.exe
- $rutaBase\sys\ssleay32.dll

- $rutaBase\sys\libeay32.dll
- $rutaBase\sys\publicKey

El siguiente esquema ilustra los archivos y estructura de directorios necesaria:

```
$rutaBase\
      |---------\exigGeneral.xml
      |----------\apps\
      |----------\jobs\
      |----------\results\
      |----------\sys\
                    |---\curl.exe
                    |---\ssleay32.dll
                    |---\libeay32.dll
                    |---\publicKey
```

En el subdirectorio *apps* se guardan todas las aplicaciones necesarias para que un trabajo pueda realizarse. Los trabajos descargados se guardan en la carpeta *jobs* mientras que los resultados generados listos para subir al servidor Web se almacenan en la carpeta *results*. En la carpeta *sys* se guardan sólo los archivos indicados. El programa *curl.exe* necesita de los archivos *ssleay32.dll* y *libeay32.dll* para ejecutarse correctamente. El archivo *publicKey* es una clave pública en formato PEM y es usada para verificar la validez de una firma digital. El archivo *exigGeneral.xml* es un documento en formato XML como el siguiente:

```
<?xml version="1.0" standalone="yes" ?>
<ÉxigusInternalData>
    <Worker version="1.0.00" id="idWorker001" />
    <cpu name="AMD Athlon(TM) XP2000+" speedMhz="1678" />
    <so name="Windows 2000" version="5.0" />
    <ram sizeMB="191" />
    <hd freeMB="300" />
    <Master ip="138.466.12.1" domain="Êxigus.com" port="7154" name="Êxigus Master" />
    <Jobs totalDone="385" leftNow="13" actualId="qfs00004" />
    <Tasks totalDone="372" leftNow="0" actualId="x" />
    <Activity actual="realizando trabajos" />
</ÉxigusInternalData>
```

Los elementos *Worker*, *cpu*, *so*, *ram*, *hd* en conjunto indican la plataforma donde se ejecuta el trabajador. Estos elementos se actualizan cada vez que inicia la ejecución del trabajador. El elemento Master tiene los atributos *ip*, *domain*, *port* y *name*. En el atributo *ip* se especifica la dirección IP del maestro Éxigus al cual se le hacen las solicitudes. En caso de no conocer la dirección IP, se puede indicar el dominio usando el campo correspondiente. El campo *port* señala cual es el puerto en el que el maestro estará atendiendo las solicitudes. Por último, el atributo *name* sólo sirve con fines informativos.
Los elementos *Jobs*, *Tasks* y *Activity* sirven para registrar la estadística de los trabajos realizados, además de informar la actividad actual del trabajador. Estos campos se actualizan conforme transcurre la ejecución del trabajador.

3.4.2 Descripción del flujo de ejecución

Una vez que se ha visto la estructura de directorios y archivos necesarios, se muestra el orden y la forma en que se ejecutan las funciones que conforman el trabajador. Cabe señalar que no se indican los parámetros exactos que toman las funciones, por cuestiones de simplicidad. Si se desean conocer los parámetros y los valores de regreso para cada función mencionada, se debe revisar el código fuente o la documentación del mismo.

Todo empieza en el archivo *exigWorker.cpp*, cuando algún programa manda llamar a la función *EXIGWORKER_init( )*. Esta función no toma parámetros y regresa un 0 para indicar que no hubo problemas o un valor distinto en caso contrario. Esta función primero trata de abrir o crear, en caso de no existir, 3 eventos distintos. Estos eventos sirven para indicarle a la función *EXIGWOKER_MainThread( )*, definida en el mismo archivo si debe detener sus operaciones un momento, reanudarlas o terminarlas definitivamente. Una vez creados los eventos, se crea un hilo en donde se ejecutará *EXIGWOKER_MainThread( )* hasta que reciba la indicación de terminar, por medio de un evento.

EXIGWOKER_MainThread( ) es la función que lleva el control de la máquina de estados y determina los puntos principales de ejecución. Lo primero que hace es actualizar el contenido de *exigGeneral.xml* por medio de una llamada a *update_exigGeneral_host_info( )*, definida en el archivo *ExigVar.cpp*, con la información de hardware y software del equipo donde se está ejecutando. A continuación determina donde se encuentra el directorio que

tomará como ruta base para saber la localización absoluta de los archivos y subdirectorios que necesita, por medio de una llamada a la función *init_worker_paths( )*. La ruta base se lee de una clave del registro de Windows.

Los pasos anteriores se ejecutan una sola vez. A continuación se determina en que estado debe iniciar la máquina de estados del trabajador. Existen 4 estados principales y 2 especiales. Los principales son:

1. Obtener clave de trabajador
2. Obtener trabajos
3. Realizar trabajos
4. Enviar resultados

Mientras que los estados especiales son:

- Esperar
- Terminar

La primera vez que se ejecuta el trabajador en un equipo, sigue la secuencia de estados principales `1->2->3->4->2->3...`, es decir, la clave de trabajador se obtiene una sola vez, mientras que obtener trabajos, realizarlos y enviar resultados se repite en secuencia múltiples veces. Al finalizar un estado principal se puede pasar al siguiente en la secuencia o a uno de los estados especiales, dependiendo, por ejemplo, si no hay acceso a la red de comunicaciones o se registra un evento indicando terminar.

La forma de determinar en cual estado iniciar es bastante simple:

1.- Si no existe un valor para el atributo *id* del elemento *Worker* en el archivo *exigGeneral.xml*, el estado de inicio debe ser obtener clave de trabajador, en caso contrario hay que ir al paso 2.

2.- Se revisa el subdirectorio *$rutaBase\jobs* y se determina si existe un subdirectorio. Si existe alguno entonces se debe iniciar en el estado realizar trabajos. En caso contrario hay que ir al paso 3.

3.- Se buscan archivos Zip en *$rutaBase\jobs*. Si existe al menos un archivo Zip, el estado de inicio es realizar trabajos. En caso contrario hay que ir al paso 4.

4.-Revisar la existencia de archivos con extensión *.up* en *$rutaBase\results*. Si existe al menos uno, el estado de inicio es enviar resultados. En caso contrario el estado de inicio es obtener trabajos.

Los estados especiales se atienden sin hacer uso de funciones especializadas. En cambio, para los estados generales existe una función principal que se encarga de realizar lo que indica el estado. Estas funciones son *get_idworker( )*, definida en *getJob.cpp*; *get_job_from_master( )*, también definida en el archivo anterior; *doJobs( )* definida en *doJob.cpp* y por último *send_results( )* en *sendResults.cpp*. Las funciones indicadas anteriormente atienden los estados obtener clave de trabajador, obtener trabajos, realizar trabajos y enviar resultados respectivamente.

Para obtener una clave de trabajador, desde *EXIGWOKER_MainThread()* se llama a la función *get_idworker( )*. Esta función prepara un mensaje *idWorkerRequest* (véase la documentación relativa al diseño). Del archivo *exigGeneral.xml* determina la dirección IP o dominio, así como el puerto del maestro al cual enviarle sus solicitudes. Con estos datos abre un socket de comunicación, envía el mensaje y espera por un mensaje de respuesta del maestro. Cualquier mensaje de respuesta recibido pasa primero por *verifySignMessage( )* definida en *digitalSign.cpp*. Esta última función determina si la firma de un mensaje es válida o no. En caso de no ser válida, simplemente se desecha la respuesta y la función *get_idworker( )* termina. En caso de que la firma sea válida, se determina el tipo de respuesta, la cual puede ser: a)que el maestro asignó una nueva clave de trabajador, b)el maestro indicó que hay que esperar cierto tiempo antes de otra solicitud o, c)el maestro indicó que se debe cambiar de maestro. En cuanto se recibe el mensaje la función *get_idworker()* termina y el mensaje recibido se pasa a la función *attend_getIdWorker_resolution( )* definida en *exigWorker.cpp*, la cual determina si repite el estado actual (en el caso de recibir un mensaje espera o cambiar de maestro) o se pasa al estado obtener trabajos.

Para obtener trabajos, desde *EXIGWOKER_MainThread( )* se llama a la función *get_job_from_master( )*. Esta función prepara un mensaje *jobRequest*, en el cual se indica la

plataforma de hardware y software donde se está ejecutando actualmente el trabajador. El mensaje preparado se envía y se espera una respuesta del maestro.

Se valida la firma digital de la respuesta y si se aprueba se determina el tipo de respuesta. Las posibles respuestas son: descargar trabajos, actualizar aplicaciones, esperar o cambiar de maestro. En el caso de estas dos últimas se procede de manera similar al estado obtener clave de trabajador. Si la respuesta del maestro indica que hay descargar trabajos se llama a la función *get_jobs( )* definida en *getJob.cpp*. Si la respuesta del maestro es actualizar aplicaciones, se llama a la función *worker_update( )* definida también en *getJob.cpp*. En cualquiera de estos dos casos, el objetivo es descargar los archivos indicados por el maestro de un servidor Web. La diferencia consiste que para los trabajos, los archivos se guardan en el subdirectorio *$rutaBase\jobs*, mientras que se usa *$rutaBase\apps* para las aplicaciones. Cuando un archivo ha sido descargado, se llama a la función *verifySignFile( ),* definida en *digitalSign.cpp,* la cual determina si el archivo sufrió o no alteraciones en su transmisión. Si la firma digital no es válida, se reintenta obtener el archivo hasta que la firma sea correcta. Una vez que todos los archivos se han descargado, la función *get_job_from_master( )* termina. Si lo último que se obtuvieron fueron aplicaciones, la máquina de estados vuelve a llamar a la función *get_job_from_master( )* para ahora sí obtener trabajos.

Para realizar los trabajos, desde *EXIGWOKER_MainThread( )* se llama a la función *doJobs( ).* Esta función lo primero que hace es revisar si existe un subdirectorio (cualquiera) en el directorio *$rutaBase\jobs*. La existencia de un subdirectorio quiere decir que un trabajo en formato Zip fue descomprimido a ese directorio, junto con las aplicaciones que necesita. Si no existe ningún subdirectorio, se toma el primer archivo Zip que exista en *$rutaBase\jobs* y se descomprime en una subcarpeta llamada como el nombre del archivo, junto con las aplicaciones que necesita para realizar el trabajo. Una vez que existe el subdirectorio, se revisa en él por la existencia de un archivo denominado *progress.txt.* En este archivo se lleva el progreso en la ejecución de cada una de las tareas que conforman el trabajo. Para determinar cuales tareas hay que realizar, se analiza la especificación de tarea. Una vez que todas las tareas se han terminado (ya sea bien o mal), se empacan los resultados según lo indica la sección *Results* en la especificación de la tarea. Al mismo tiempo se determina la dirección del servidor Web al que hay que subir el archivo con los resultados y se crea un archivo denominado igual que el archivo de resultado, pero con extensión *.up.* Ambos archivos se copian al directorio *$rutaBase\results* y la carpeta

donde se realizó el trabajo se borra, así como el archivo que lo originó. El proceso se repite por cada archivo Zip que exista en el directorio *$rutaBase\jobs*, hasta que no haya más archivos, con lo cual la función *doJobs( )* termina y la máquina de estados puede pasar al estado enviar resultados.

Para enviar los resultados, desde *EXIGWOKER_MainThread( )* se llama a la función *send_results( )*. Esta función crea un archivo denominado *sends.txt*, y en él se indica la dirección url hacia donde se debe subir cada uno de los archivos Zip que se encuentran en el directorio *$rutaBase\results*. Una vez que el archivo *sends.txt* ha sido creado, comienza un ciclo que termina hasta que todos los archivos se hayan subido al servidor Web. Cuando el ciclo termina, se borra todo el contenido del directorio *$rutaBase\results* para que la máquina de estados pueda pasar al estado de obtener trabajos.

3.5 IMPLEMENTACIÓN DEL MAESTRO

El maestro Éxigus está conformado por 3 programas: un servidor Web, un administrador de bases de datos y un programa denominado Coordinador, el cual se encarga de manejar a los programas anteriores, así como tomar decisiones respecto los trabajos asignados y los resultados recibidos.

3.5.1 Implementación de la base de datos

Para llevar control de la información proporcionada y generada, se hace uso de una base de datos. Esta base de datos, denominada *exig2* fue implementada usando el administrador de base de datos MySQL. La base de datos está conformada por 8 tablas y la descripción de las mismas, así como sus campos se muestran a continuación:

1. Tabla ***workers***

La función de esta tabla es llevar control de los trabajadores registrados en el sistema.

Campo	Tipo de dato	Comentario
idWorker	Caracter con 64 de ellos, clave primaria	En este campo están registradas las claves de trabajador que se manejan en el sistema

idHwSw	Entero, clave foránea	Esta es la clave del ambiente de hardware y software donde se ejecuta el trabajador
timeIdAsign	Fecha y hora	Momento en el que se asignó la clave de trabajador
flags	Caracter, con 10 de ellos	Si el valor es 's', quiere decir que el trabajador es un saboteador

2. Tabla *hwSwProfile*

En esta tabla se indican las características de hardware y software donde se ejecutan los trabajadores.

Campo	Tipo de dato	Comentario
idHwSw	Entero, clave primaria	En este campo están registradas las claves que identifican el ambiente de hardware y software donde se ejecuta el trabajador
version	Caracter (32 de ellos)	Cadena que identifica la versión del trabajador
osName	Caracter (32 de ellos)	Cadena que identifica el nombre del sistema operativo
osVersion	Caracter (32 de ellos)	Cadena que identifica la versión del sistema operativo
cpuName	Caracter (32 de ellos)	Cadena que identifica el nombre del procesador
cpuSpeedMhz	Entero	Valor que indica la velocidad en Mhz del procesador
ramSizeMB	Entero	Valor que indica la memoria RAM física, dada en Megabytes, con que cuenta el trabajador
hdFreeMB	Entero	Valor que indica el espacio disponible en disco duro con que cuenta el trabajador
ip	Caracter (32 de ellos)	Cadena que indica la dirección ip y el puerto del trabajador

3. Tabla *jobProfiles*

En esta tabla se indican los requerimientos de hardware para que un trabajo pueda ser asignado a un trabajador.

Campo	Tipo de dato	Comentario
idJobProfile	Entero, clave primaria	En este campo están registradas las claves de los perfiles de hardware para los trabajos
minCpuMhz	Entero	Velocidad mínima en Mhz del CPU del trabajador para que el trabajo que use este perfil pueda ser asignado.
maxCpuMhz	Entero	Velocidad máxima en Mhz del CPU del trabajador para que el trabajo que use este perfil pueda ser asignado.
minRamMB	Entero	Memoria RAM física mínima, expresada en megabytes, que debe poseer el equipo donde se ejecuta el trabajador para que el trabajo que use este perfil pueda ser asignado.
minHDFreeMB	Entero	Espacio libre en disco duro mínimo, expresado en megabytes, que debe poseer el equipo donde se ejecuta el trabajador para que el trabajo que use este perfil pueda ser asignado.

4. Tabla *jobs*

En esta tabla se registran los trabajos que se asignarán a los trabajadores.

Campo	Tipo de dato	Comentario
idJob	Caracter, con 64 de ellos, clave primaria	En este campo están registradas las claves de los diversos trabajos por asignar.
idJobProfile	Entero, clave foránea	Esta es la clave del perfil del trabajo que se toma en cuenta antes de asignarlo a un trabajador.

fileName	Caracter(64 de ellos)	Nombre y extensión del archivo que contiene el trabajo. No se incluyen rutas.
flags	Caracter(10 de ellos)	Los posibles valores son: 'r' para indicar que el trabajo está resuelto 's' para indicar que es un trabajo inspector (el campo vacío indica que el trabajo no ha sido resuelto)
result	Caracter(45 de ellos)	Resumen de tipo SHA1 de todos los archivos contenidos en un resultado para un trabajo inspector. Este campo se usa en combinación con flags = 's'.

5. Tabla *Asigns*

En esta tabla se registran las asignaciones de los trabajos.

Campo	Tipo de dato	Comentario
idAsign	Caracter, con 64 de ellos, clave primaria	En este campo están registradas las claves de las asignaciones de los trabajos
idWorker	Caracter, con 64 de ellos, clave foránea	Se registra la clave del trabajador al cual se le asignó un trabajo.
idJob	Caracter, con 64 de ellos, clave foránea	Clave del trabajo asignado
asignTime	Fecha y hora	Momento en que se asignó un trabajo

6. Tabla *results*

En esta tabla se registran las recepciones de resultados para los trabajos asignados.

Campo	Tipo de dato	Comentario
idResult	Entero, clave primaria	En este campo están registradas las claves de las recepciones de los resultados.
idAsign	Entero, clave foránea	Clave de la asignación correspondiente al resultado recibido
idWorker	Caracter, con 64 de ellos, clave foránea	Clave del trabajador que envía los resultados
idJob	Caracter, con 64 de ellos, clave foránea	Clave del trabajo para el cual se reciben los resultados
result	Caracter, con 45 de ellos	Resumen de tipo SHA1 de todos los archivos contenidos en un resultado para cualquier trabajo
fileName	Caracter, con 255 de ellos	Nombre y extensión del archivo que contiene el resultado. No se incluyen rutas.
receiveTime	Fecha y hora	Momento en que se recibió el resultado
flags	Caracter, con 10 de ellos	Los posibles valores son: 'n','v', 'p', 'c' para indicar que el resultado ha sido verificado usando algún método de tolerancia a fallos. 'w' para indicar la recepción ficticia de un resultado correcto para un trabajo descubridor. Es usado junto con el método de tolerancia a fallos 'credibilidad' para recompensar a un trabajador. (el campo vacío indica que se ha recibido el resultado, más no ha sido verificado)
report	Caracter, con un número indefinido de ellos	Este campo contiene el texto del informe de resultados generado por un trabajador para un trabajo dado.

7. Tabla *proyectVars*

En esta tabla se maneja sólo un registro que contiene información que controla aspectos de recepción y asignación.

Campo	Tipo de dato	Comentario
faultMethod	Entero	Indica el método de tolerancia a fallos usado para asignar trabajos y registrar resultados como válidos. Los valores posibles son: 0 sin tolerancia a fallos 1 voto mayoritario 2 inspección al azar 3 credibilidad
mVote	Entero	Si el método de tolerancia a fallos es voto mayoritario, este campo indica cual es el número mínimo de resultados iguales que se deben recibir para un mismo trabajo, para que éste sea marcado como resuelto.
qRate	Decimal	Si el método de tolerancia a fallos es credibilidad o inspección al azar, este campo indica la tasa a la cual deben enviarse los trabajos especiales a los trabajadores.
fFraction	Decimal	Si el método de tolerancia a fallos es credibilidad, este campo indica el porcentaje dividido entre 100 del numero máximo de trabajadores considerados como saboteadores
CrW	Decimal	Si el método de tolerancia a fallos es credibilidad, este campo indica el umbral de credibilidad mínimo que deben alcanzar los resultados para marcar los correspondientes trabajos como resueltos
minSecsToNewRequests	Entero	Tiempo mínimo en segundos que debe esperar un trabajador al que recién se le asignó una clave de trabajador para que pueda solicitar trabajos
minSecsToJobAsign	Entero	Tiempo mínimo en segundos que debe esperar un trabajador después de su última asignación de trabajos o recepción de resultados, para que se le asigne un nuevo trabajo
minSecsToResultReceive	Entero	Tiempo mínimo en segundos que debe esperar un trabajador después de su última asignación de trabajos o recepción de resultados, para que se le acepte un nuevo resultado
maxJobsToAsign	Entero	Número máximo de trabajos que se asignarán a un trabajador, si es que hay suficiente disponibles
resultsReceivePath	Caracter, ilimitado	Ruta absoluta donde se guardaran los resultados recibidos. Ejemplo: c:\proyecto\resultados\
urlJobPrefix	Caracter, ilimitado	Prefijo que se coloca antes del nombre del trabajo para formar la ruta completa para descargarlo. Ejemplo: Nombre trabajo: trabajo001.zip urlJobPrefix: http://master.com/jobs/ ruta completa: http://master.com/jobs/trabajo001.zip
urlAppPrefix	Caracter, ilimitado	Prefijo que se coloca antes del nombre de una aplicación para formar la ruta completa para descargarlo. Ejemplo: Nombre aplicación: render.zip urlAppPrefix: http://master.com/apps/ ruta completa: http://master.com/apps/render.zip
appsList	Caracter, ilimitado	Lista de aplicaciones que los trabajadores deben tener para poder realizar cualquier trabajo que se les asignen.

	La lista es un conjunto de nombres cortos de archivo separados por un espacio. Ejemplo: "render.zip tga2jpg.zip jpg2mpg.zip"

8. Tabla *saboteurInfo*

En esta tabla se pasan todos los resultados que generó un trabajador y que ya no son tomados en cuenta debido a que el trabajador falló en un trabajo inspector.

Campo	Tipo de dato	Comentario
idSaboteur	Entero, clave primaria	En este campo están registradas las claves para cada resultado generado por un trabajador saboteador
idWorker	Caracter, con 64 de ellos, clave foránea	Clave del trabajador saboteador
idJob	Caracter, con 64 de ellos, clave foránea	Clave del trabajo que realizó el trabajador
result	Caracter, con 45 de ellos	Resumen de tipo SHA1 de todos los archivos contenidos en un resultado para cualquier trabajo
fileName	Caracter, con 255 de ellos	Nombre y extensión del archivo que contiene el resultado. No se incluyen rutas.
asignTime	Fecha y hora	Momento en que se asignó el resultado
receiveTime	Fecha y hora	Momento en que se recibió el resultado

Por último, en la figura 3.15 se muestra el diagrama Entidad-Relación.

3.5.2 Implementación del servidor Web

En sí el servidor Web no requiere gran configuración. Sólo tiene dos funciones: servir los trabajos y las aplicaciones para que puedan ser descargadas y guardar los resultados que se suban hacia él.

La primera función es tan simple como colocar los archivos de trabajos y aplicaciones en un directorio al cual tenga acceso el servidor Web. Por ejemplo, en la plataforma Windows, el servidor Web Apache se instala por omisión en el directorio *C:\Archivos de programa\Apache Group\Apache* y todos los archivos los trata de leer tomando como ruta base *C:\Archivos de programa\Apache Group\Apache\htdocs*. De esta forma, un servidor Web instalado en ese directorio, con un dominio denominado *servidorpc.net* y con la configuración por omisión, si se le solicitara el archivo *http://servidorpc.net/informes/ventas.html*, el servidor regresaría el contenido del archivo *C:\Archivos de programa\Apache Group\Apache\htdocs\informes\ventas.html*

Este mismo ejemplo funciona para cualquier grupo de archivos, como por ejemplo trabajos, a los cuales pueda tener acceso el servidor Web.

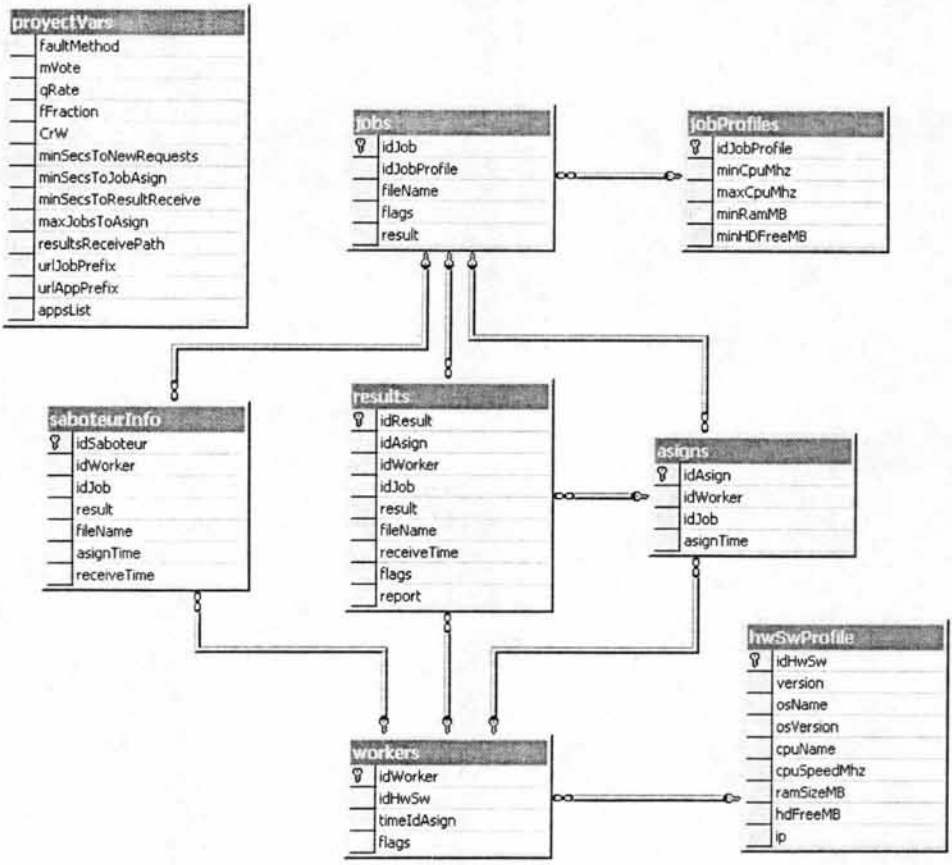

figura 3.15 Diagrama E-R para la base de datos del maestro Éxigus

La segunda función, la de recibir los resultados de los trabajadores, es un poco más complicada. Dentro del protocolo http no existe una forma directa y sencilla para subir archivos a un servidor Web, como si lo es para descargarlos. En este caso se requiere de un programa auxiliar que se encargue de esa función, utilizando CGI (Common Gateway Interface). CGI define una manera por medio de la cual un servidor Web interactúa con programas externos. Durante una transacción CGI, el servidor y un cliente hacen disponibles unas variables, de modo que pueden comunicarse entre ellos. Estas variables indican el tipo de navegador, el tipo de servidor, el nombre del programa externo por ejecutar, etc. En el protocolo http existe la operación POST, por medio de la cual un cliente envía información a un servidor Web y éste se la pasa a un programa externo, por medio de su propia salida estándar (stdout) hacia la entrada estándar (stdin) del programa externo.

La información se empaca de una manera especial por lo que el programa externo puede desempacarlo, interpretarlo y tomar las acciones correspondientes.

Tomando en cuenta lo anterior, se creó un programa en lenguaje C, capaz de interactuar con el servidor Web usando CGI, denominado *resrecv.exe*. Este programa espera a que un cliente realice una petición de subir resultados hacia *http://nombreServidor/cgi-bin/rescvres.exe*. En ese momento, el servidor Web le pasa los datos que reciba del cliente hacia el programa CGI.

En concreto, el programa CGI *resrecv.exe* espera leer 3 variables: la clave del trabajador, la clave del trabajo al que corresponde el resultado y el archivo de resultados en si. Si toda esta información está completa se proceden a revisar las siguientes condiciones, interactuando para ello con la base de datos:

1. El trabajador (indicado por la clave) debe estar registrado
2. El trabajador no debe ser saboteador
3. El resultado debe corresponder a un trabajo asignado
4. Se debe cumplir el tiempo mínimo para recibir resultados después de la última asignación de trabajos.
5. El archivo con los resultados no debe exceder un límite de 2 Megabytes.

Si todas las condiciones anteriores se cumplen, el programa CGI, *rescvres.exe*, guarda el archivo de resultados en el directorio especificado por el campo *resultsReceivePath*, de la tabla *proyectVars*. En el caso de que alguna de las condicione falle, la conexión termina inmediatamente.

Puede darse el caso de recibir más de una copia del mismo resultado, por lo que el programa CGI los renombra para evitar sobreescrituras, agregando un número al final del nombre del archivo, acorde con el número de la copia. Por ejemplo, si se recibe el archivo de resultados *resABC.zip* y ya existe uno con el mismo nombre en el directorio destino, se renombra el nuevo archivo a *resABC.zip.001*.

Una vez que el archivo ha sido guardado en disco duro, se registra su recepción en la base de datos, en concreto, en la tabla *results*. Todos los campos de la tabla, excepto *result*, *flags* y *report* son llenan con los valores adecuados para el nuevo registro. La razón por la cual *result* no es llenado es debido a que descomprimir un archivo y calcular el SHA1 de los archivos que contiene puede ser tardado de realizarlo al momento, lo cual puede afectar el número de resultados por segundo que se puedan recibir.

Existe otro programa que se encarga de monitorear por nuevos resultados recibidos, el cual llena los campos que no hizo el programa CGI, en un momento más adecuado y el cual está descrito en la siguiente componente que forma el maestro Éxigus: el Coordinador.

3.5.3 Implementación del Coordinador

El Coordinador se implementó creando varios archivos de código fuente escritos en lenguaje Java, haciendo uso de clases definidas para la versión 1.4. Esto en cierto modo garantiza un correcto funcionamiento en plataformas para las cuales exista una máquina virtual de Java, como pueden ser Windows, GNU/Linux, FreeBSD y Solaris entre otros. Debido a que rutinas para conexión a base de datos, manejo de documentos XML y creación de firmas digitales tienen una implementación deficiente o en otros casos inexistente dentro de las bibliotecas principales de Java, se tuvo que recurrir a bibliotecas externas. Estas bibliotecas son NanoXML (para documentos XML), y BouncyCastle (para el manejo de firmas digitales usando cifrado RSA), junto con el conector MySQLJDBC para realizar el enlace con la base de datos.

El Coordinador, aparte de llevar el control parcial del servidor Web y del administrador de base de datos, agrega funcionalidad adicional que permite la correcta interacción entre los trabajadores y el maestro en sí. Todas las funciones del Coordinador se pueden dividir en dos grupos: aquellas que intervienen en la atención de solicitudes de los trabajadores y las que están involucradas con el monitoreo y manejo de los resultados.

Atención de solicitudes. Básicamente existen dos tipos de solicitudes que puede hacer un trabajador: 1) solicitar una clave de trabajador y 2) solicitar trabajos. En ambos casos, el trabajador debe preparar un mensaje adecuado y abrir una conexión hacia la dirección IP y el puerto específico del maestro. La clase *ExigRequestProcessor* se encarga de mantener abiertas hasta 8 conexiones realizadas hacia el puerto de atención del Maestro. Cuando detecta un intento de conexión, y si no se han rebasado el límite de conexiones, se crea un hilo que se encarga de atender esa conexión. La primera función del nuevo hilo es determinar el tipo de solicitud que el trabajador está realizando. Para ello llama al método *get_message_type()* definido en la misma clase *ExigRequestProcessor*. Dependiendo del tipo de solicitud, se manda llamar al método *attend_idWorker_request()* o *attend_job_request()*,

para la atender solicitudes de clave de trabajador o trabajos, respectivamente. Ambos métodos están definidos en la misma clase *ExigRequestProcessor.*

El método *attend_idWorker_request*(), trata de obtener una clave de trabajador disponible en la tabla *workers* de la base de datos. Para ello llama al método *getIdWorker()* de una instancia de clase de *ExigIdWorkerProcessor*, definida en el archivo fuente con el mismo nombre. El método *getIdWorker( )* tiene un seguro para evitar que dos o más hilos intenten obtener la misma clave de trabajador al mismo tiempo. Si existe una clave disponible en la tabla *workers*, ésta se marca como asignada y se regresa a quien llamó a la función. En caso de no existir, simplemente se regresa como clave de trabajador una cadena vacía. Entonces el método *attend_idWorker_request*(), crea un mensaje de asignación de clave en caso de haber una disponible o un mensaje de espera (en caso de no haber claves disponibles). El mensaje es firmado digitalmente llamando al método *signMessage( )* de una instancia de clase *ExigSignMessage*, definida en el archivo con el mismo nombre. El mensaje firmado es entonces enviado por el canal de comunicación que abrió el trabajador y el hilo que atendió la solicitud termina su labor.

El otro tipo de solicitud, la de trabajos, se atiende llamando al método *attend_idWorker_request( ),* de una instancia de clase de *ExigRequestProcessor.* Lo primero que realiza este método es actualizar la información del hardware y software donde se ejecuta el trabajador, la cual va incluida en el mensaje de solicitud de trabajo. Para ello se llama al método *updateWorkerDBHwProfile( )* de la clase *ExigWorkerUpdate*, definida en el archivo del mismo nombre. A continuación se determina si el trabajador cuenta con las aplicaciones necesarias para realizar los posibles trabajos que se le puedan asignar. Para ello se llama al método *getWorkerUpdateMessage( )* de la clase *ExigWorkerUpdate.* El método compara la lista de aplicaciones necesarias indicadas en el campo *appsList* de la tabla *proyectVars* de la BD, con la lista de aplicaciones que indica el trabajador en su mensaje de solicitud de trabajos. Si al trabajador le hacen falta aplicaciones, se crea un mensaje indicando cuales debe descargar. El mensaje se firma, se envía y termina la conexión.

En caso de que el trabajador cuente con las aplicaciones necesarias, se procede a llamar al método *getJobAsignMessage( )* de la clase *ExigJobAsign.* Este método regresa una cadena con el mensaje indicando los trabajos que debe realizar el trabajador o una cadena vacía para señalar que por algún motivo no se le asignarán trabajos al trabajador. Los motivos

pueden ser: 1) el trabajador es un saboteador, 2) el trabajador no cumple con los tiempos mínimos de registro/asignación, 3) no existen trabajos disponibles que cumplen con los requerimientos de hardware y 4) todos los trabajos han sido resueltos satisfactoriamente. Si *getJobAsignMessage( )* regresa un mensaje de asignación de trabajos, se firma y se envía al trabajador como respuesta. Si no existe un mensaje de asignación de trabajos, se crea un mensaje de espera, se firma, se envía y en cualquier caso, termina la conexión.

Realmente el punto interesante en la solicitud de trabajos es determinar los trabajos que se asignarán. Partiendo del método *getJobAsignMessage( )*, éste llama primero al método *isWorkerValid( )* de la clase *ExigWorkerValidity*, el cual determina si el trabajador es saboteador o no. Asumiendo que el trabajador no sea saboteador, se llama al método *canAcceptRequests( )* de la clase *ExigDoSValidity*, el cual indica si se cumplen con los tiempos mínimos de registro/asignación. Si los tiempos son aceptables, se determina la categoría de trabajos y el número máximo que se le pueden asignar al trabajador solicitante, llamando a *getMaxJobsToAsign( )* y a *getIdJobProfile( )*, los cuales son métodos de *ExigJobAsign*. A continuación se llama al método *getIdJobsToAsign( )* de la clase *ExigJobQueue*. Este último método tiene un seguro para evitar que se rompa la secuencia de asignación de trabajos en caso de que dos o más hilos intenten llamarlo al mismo tiempo. En la clase *ExigJobQueue* se mantienen varias colas circulares (una por cada perfil de trabajo) con los trabajos que todavía no han sido resueltos. Cuando se llama al método *getIdJobsToAsign( )* se le pasan como parámetros el perfil y el número máximo de trabajos por obtener. El método determina cuales trabajos se pueden asignar y el apuntador de la cola circular cambia a los siguientes trabajos disponibles. Antes de regresar las claves de los trabajos por asignar, se revisa si hay que agregar o no algún trabajo inspector. Esto sólo se realiza si existe alguno disponible y si el método de tolerancia a fallos indicado en el campo *faultMethod* de la tabla *proyectVars* de la BD tiene valor de 2 ó 3 (Inspección al azar o Credibilidad, respectivamente). Dependiendo de la tasa de asignación de trabajos descubridores (campo *qRate* de la tabla *proyectVars*) se determina si hay que agregar o no un trabajo inspector, seleccionado aleatoriamente, a la lista de trabajos por asignar previamente determinada. Finalmente, con la lista de las claves de trabajos por asignar, se prepara un mensaje señalando las direcciones URL de donde deben ser descargados y se le envía el mensaje firmado al trabajador.

Monitoreo y manejo de los resultados. El manejo de los resultados se hace en parte en el programa CGI creado para el servidor Web. Sin embargo, ese programa sólo los registraba e

incluso dejaba algunos campos incompletos en el registro correspondiente en la tabla *results* de la BD. De hecho, tampoco se validaban los resultados de acuerdo a algún método de tolerancia a fallos. Debido a esto, la otra parte que conforma la funcionalidad del Coordinador se encuentra en el monitoreo y manejo de los resultados. A diferencia de la atención de solicitudes, este código tiene un flujo secuencial que no hace uso de hilos adicionales.

Cuando el Coordinador inicia (el punto de inicio del Coordinador se encuentra en la clase *ExigMasterJava*, definida en el archivo del mismo nombre), se crean dos hilos para ejecutar dos clases simultáneamente. Una clase atiene las solicitudes (*ExigRequestProcessor*) y otra monitorea y maneja los resultados (*ExigMonitor*, definida en el archivo fuente del mismo nombre.)

Regresando al manejo de resultados, cuando se inicia la clase *ExigMonitor* se ejecuta un solo método que entra en un ciclo permanente. En este ciclo lo primero que se realiza es revisar si se han recibido nuevos resultados (los que ha registrado el programa CGI). Para ello se llama al método *thereAreNewResults( )* de la clase *ExigMonitorUtils*. Este método busca los registros en la tabla *results* de la BD que en su campo de *flags* no tengan ningún valor. Si existen nuevos resultados, se llama entonces al método *insertNewResultsReportAndSha1InDB( )* de la misma clase *ExigMonitorUtils*. Este método realiza tres actividades:

1. Revisa que el archivo de resultados indicado en el registro correspondiente no esté corrupto, lo que en caso de darse obliga a que el resultado sea eliminado del sistema.
2. Extrae el texto del archivo *jobReport.rpt*, contenido en todos los archivos de resultados, y lo guarda en el campo *report* para el registro correspondiente en la tabla *results* de la BD.
3. Calcula el SHA1 conjunto de todos los archivos contenidos en el archivo Zip del resultado (todos excepto *jobReport.rpt*)

Una vez realizadas estas actividades, se determina la validez de los nuevos resultados recibidos, dependiendo del método de tolerancia a fallos, y se actúa consecuentemente. Si no existen nuevos resultados o todos los trabajos ya han sido marcados como resueltos, el ciclo permanente entra a un estado momentáneo de espera, para comenzar de nuevo el monitoreo.

Para manejar los nuevos resultados, cuando no hay tolerancia a fallos (*faultMethod = 0* en *proyectVars* de la BD), se llama al método *newResultsManagement( )* de la clase *ExigNoFaultTolerance*. Este método marca como resuelto el trabajo correspondiente al nuevo resultado, si es que el trabajo todavía no se marca como resuelto. Después se marcan los nuevos resultados con una 'n' en su campo de *flags* en la tabla *results* para evitar caer en un ciclo infinito considerando como nuevos resultados los ya procesados.

Cuando el método de tolerancia a fallos es Voto mayoritario (*faultMethod = 1* en *proyectVars* de la BD), se llama al método *newResultsManagement( )* de la clase *ExigMajorityVoting*. Este método busca si con los nuevos resultados recibidos se alcanza la mayoría de votos para un trabajo (esto es, que los resultados para un mismo trabajo sean iguales en un determinado número indicado en el campo *mVote* de *proyectVars*). Si se alcanza la mayoría de votos el trabajo correspondiente se marca como resuelto. Posteriormente se marcan los resultados con una 'v' en su campo de *flags* en la tabla *results* para indicar que no deben considerarse como nuevos resultados.

En el caso de que el método de tolerancia a fallos sea Inspección al azar (*faultMethod = 2* en *proyectVars* de la BD), se llama al método *newResultsManagement( )* de la clase *ExigSpotChecking*. Este método realiza una de dos acciones dependiendo si el nuevo resultado viene de un trabajo inspector o no. Si el resultado no tiene que ver con trabajos inspectores, simplemente se marca como resuelto el trabajo correspondiente. Sin embargo, si el resultado nuevo tiene que ver con un trabajo inspector, se revisa si el resultado es igual al esperado. En caso de ser iguales, no se toma ninguna acción, en cambio si son diferentes, se invalidan todos los resultados recibidos por el trabajador que falló en el trabajo inspector y el trabajador correspondiente se marca como saboteador. Después se marcan los nuevos resultados con una 'p' en su campo de *flags* en la tabla *results* para indicar que no deben considerarse como nuevos resultados.

Por último, si el método de tolerancia a fallos es Credibilidad (*faultMethod = 3* en *proyectVars* de la BD), se llama al método *newResultsManagement( )* de la clase *ExigCredibility*. Este método se comporta como el caso de trabajos descubridores, pero con una salvedad: los resultados nuevos para trabajos normales no marcan el trabajo correspondiente como resuelto a menos que se haya alcanzado el umbral de credibilidad

(*CrW* en *proyectVars*). Salvo por ese detalle, el comportamiento y las acciones punitivas son similares al método de tolerancia a fallos de trabajos descubridores.

3.6 CONCLUSIONES

En este capítulo se presentaron todos los elementos que fueron considerados en el desarrollo de Éxigus, para dar paso a un diseño y a partir de él, implementar toda la funcionalidad necesaria. Aunque gran parte de estos datos se muestran por medio de diversas representaciones, es hasta ver la infraestructura en funcionamiento cuando realmente se comprende mejor todo lo que conlleva. En el siguiente capítulo, se realizan pruebas reales de la infraestructura en proyectos de diversa naturaleza y así entender mejor sus capacidades.

4. PRUEBAS

Una vez concluido el diseño y la implementación de la infraestructura Éxigus, es el momento de realizar una serie de pruebas para verificar que se cumplan los objetivos propuestos y de paso, destacar las fortalezas y en especial, identificar las debilidades para así tener la información necesaria y poder tomar decisiones respecto al uso de esta infraestructura.

Esta sección contempla la implementación y pruebas de 3 proyectos distribuidos, denominados "espera", "trazado de rayos" y "factorización". El primer proyecto, "espera", consiste en realizar trabajos que no producen resultados útiles, sino más bien tardan en ejecutarse de forma individual lo mismo o casi lo mismo, independientemente del equipo. En el proyecto, "trazado de rayos", el objetivo es lograr la representación de un conjunto de escenas tridimensionales que conforman una animación, utilizando el método de trazado de rayos o raytracing en inglés. Por último, en el proyecto "factorización", la finalidad es obtener la factorización de un número grande, por el método de la Criba Cuadrática de Campo o QFS por las siglas en inglés de Quadratic Field Sieve.

Todos los proyectos se probaron primero en un mismo equipo, para posteriormente comprobar una mejoría de tiempo, al distribuir la carga entre varios equipos diferentes.

4.1 PROYECTO ESPERA

4.1.1 Introducción

El realizar un trabajo usando una infraestructura de cómputo distribuido toma diferente tiempo a realizarlo sin ésta. Esta diferencia de tiempo se debe a diversos motivos. Con el objetivo de eliminar en las mediciones factores como la carga actual de trabajo y memoria libre en el equipo donde se ejecuta el trabajador, se creó un tipo de proyecto donde los trabajos son lo más pequeños posibles y su tiempo de resolución es prácticamente independiente del equipo donde se ejecute. Una vez teniendo listo este tipo de proyecto, se pueden realizar una serie de observaciones y mediciones sobre la infraestructura Éxigus.

4.1.2 Descripción de los trabajos, aplicaciones y herramientas creadas

Aplicación. Se creó un programa denominado *espera.exe*. La finalidad de este programa es consumir los segundos indicados como parámetro sin realizar nada. Una vez consumido el tiempo, se crea un archivo denominado *tiempoEsperado.txt* en el que se escribe el tiempo que se consumió. El programa se usa invocándolo con la siguiente línea de comandos:

```
espera.exe numSegs
```

Donde:
numSegs es un valor entero mayor a 1 el cual indica el tiempo que se esperará antes de que termine la ejecución del programa.

Para su distribución, el programa *espera.exe* junto con su firma digital, *signature*, se comprimieron en un archivo Zip denominado *espera-1.00.zip*

Trabajos. Los trabajos sólo requerían que la especificación de la tarea indicara el número de segundos que se debían consumir antes que la aplicación terminara.
Nombre usado: esperaXXX.zip (donde X puede ser cualquier dígito)
Dependencias: espera-1.00.zip
Archivos que contiene: esperaXXX.xml, signature

Archivos que produce: tiempoEsperado.txt

Ejemplo de la especificación de trabajo:

```
<?xml version="1.0" standalone="yes" ?>
<Job id='jobEsperaXXX'>
  <Dependencies>
    <App>espera-1.00.zip</App>
  </Dependencies>
  <Tasks>
    <Task id='onlyTask' rightExitCode='0' retries='0' timeToRetry='0'>
       <Process>espera.exe 1</Process>
    </Task>
  </Tasks>
   <Results>
    <Package fileName='resEsperaXXX.zip' zipLevel='1'
                    upload='http://MASTER/cgi-bin/resrecv.exe' >
     <FileToPack>esperaF.txt</FileToPack>
    </Package>
   </Results>
</Job>
```

Herramientas. Se creó una herramienta, denominada waitjbmkr.exe, para crear los trabajos. El principal parámetro indicado era el número de trabajos que se deseaban crear. Por omisión todos los trabajos realizan una espera de un segundo antes de terminar. La especificación del trabajo es entonces comprimida en un archivo Zip, que posteriormente se firma, lo cual deja los trabajos listos para ser asignados.

4.1.3 Pruebas realizadas en uno y varios equipos físicos

Las pruebas se pueden dividir en dos grandes grupos: aquellas en las que varios trabajadores se ejecutaban en un solo equipo y pruebas en las que había a lo más un trabajador por equipo físico.

Se tuvo acceso a tres PCs con hardware y software diferente, conectadas por medio de una LAN Ethernet a 100 Mbits. Los diferentes equipos cuentan con las siguientes características:

1. PC con procesador AMD XP 2000+ (1.67 GHz), 512 MB RAM, Windows 2000 Professional, servidor Web Apache versión 1.3, Administrador de Base de datos MySQL versión 2.33.58, ambiente de ejecución Java (JRE, Java Runtime Environment) versión 1.4.1.

2. PC con procesador INTEL Pentium 4, 2.0 GHz, 512 MB RAM, Windows 98.

3. PC con procesador INTEL Pentium 4, 1.4 GHz, 256 MB RAM, Windows 2000 Advanced Server.

Todos los componentes que forman el maestro Éxigus fueron instalados en el equipo número 1. Este equipo fue en donde se realizaron las pruebas para varios trabajadores ejecutándose en un solo equipo.

Condiciones de ejecución de la infraestructura Éxigus: Se crearon 32 trabajos normales, más un trabajo especial (trabajo inspector) sobre el cual ya se conocía el SHA1 de su resultado. Los trabajos fueron asignados desde 1 a 8 trabajadores distintos ejecutándose en el mismo equipo y hasta 3 trabajadores en equipos distintos. Se probaron los cuatro métodos de tolerancia a fallos. Para Voto Mayoritario se determinó que se obtuvieran al menos 2 votos por cada trabajo. En el caso de Inspección al azar y Credibilidad se estableció una tasa de asignación de trabajos inspectores con valor de 0.1 (10%). Para Credibilidad se estimó una población saboteadora del 20%, aunque en realidad ninguno de los trabajadores regresó un resultado incorrecto. El umbral de credibilidad para marcar un trabajo como resuelto se estableció en valores de 0.8, 0.9 y 0.98. En todos los casos se estableció que el número máximo de trabajos por asignar fuera de 1 solo.

Resultados:

Ejecutar 32 veces el programa *espera.exe* con un parámetro de espera de 1 segundo tomo 33 segundos sin hacer uso de la infraestructura Éxigus. Usando la infraestructura, los resultados para un número variado de trabajadores y métodos de tolerancia a fallas fueron:

Un solo equipo

Trabajadores	Tiempo en segundos para varios métodos de tolerancia a fallos (un solo equipo)					
	Sin tolerancia	Voto mayoritario	Inspección al azar	Credibilidad (varios umbrales)		
				(0.8)	(0.9)	(0.98)
1	51	107	59	60	75	146
2	33	62	32	34	49	113
3	24	49	25	27	40	119
4	20	41	20	21	37	111
5	16	32	18	19	36	111
6	15	32	18	19	40	115
7	15	29	18	19	41	122
8	16	30	18	19	44	163

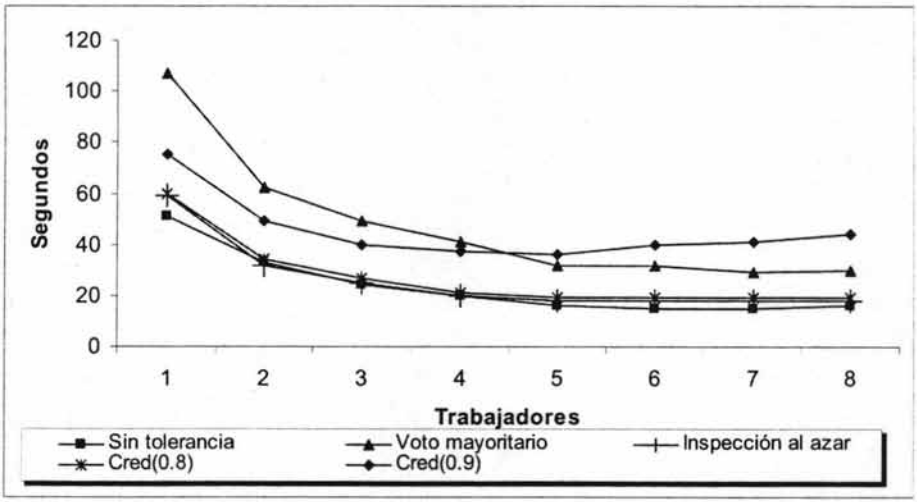

Figura 4.1 Gráfica de comparativo de tiempos para varios métodos de tolerancia a fallos en un solo equipo

Varios equipos

Trabajadores	Tiempo en segundos para varios métodos de tolerancia a fallos (varios equipos)					
	Sin tolerancia	Voto mayoritario	Inspección al azar	Credibilidad (varios umbrales)		
				(0.8)	(0.9)	(0.98)
1	51	107	59	60	75	146
2	37	66	35	37	52	118
3	23	46	25	27	42	110

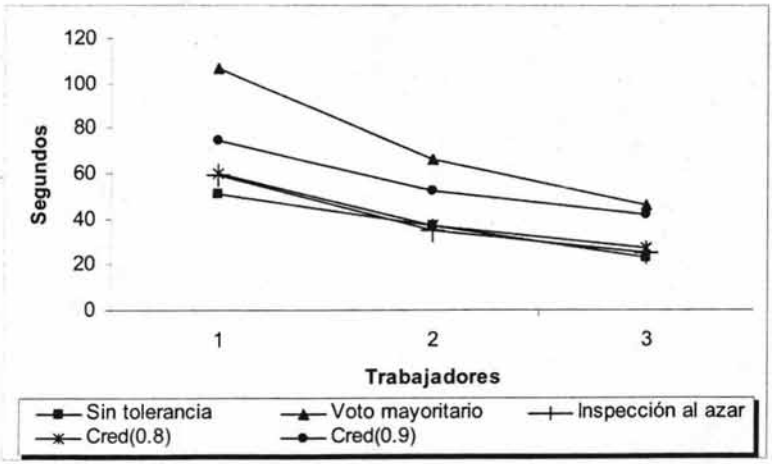

Figura 4.2 Gráfica de comparativo de tiempos para varios métodos de tolerancia a fallos en varios equipos

4.1.4 Interpretación de los resultados

En los resultados para un solo equipo, se puede notar que existen combinaciones de número de trabajadores y método de tolerancia a fallos en el cual realizar los 32 trabajos usando la infraestructura Éxigus toma menos tiempo que realizarlos de forma secuencial (por ejemplo 8 trabajadores sin tolerancia a fallos sólo tardan 16 segundos). Esto se debe a la capacidad multitarea del sistema operativo y que la aplicación *espera.exe* utiliza una cantidad mínima del tiempo del procesador cuando se ejecuta, junto con unos trabajos y resultados que ocupan poco espacio. Estas características hacen útil el proyecto Espera para estimar el comportamiento en un ambiente de ejecución formado por varios equipos físicos, tal como se pudo comprobar. En las pruebas realizadas en varios equipos, los tiempos prácticamente están escalados por un factor, debido en parte a la sobrecarga de la comunicación al enviar la información por la red física. Por lo tanto, no tiene mucho caso hacer diferencia entre las pruebas en uno y varios equipos.

Con la combinación de solamente un trabajador y sin tolerancia a fallos se puede obtener un dato muy importante: la sobrecarga que implica al sistema Éxigus asignar un trabajo y registrar un resultado independientemente del tipo de trabajo. La sobrecarga en segundos que implica manejar un trabajo para el sistema Éxigus para el sistema de pruebas anterior es de 0.5937s. Este número se obtuvo restando a el tiempo que toma resolver los trabajos (51 seg.) el tiempo que tomarían en realizarse de forma secuencial (33 seg.) y dividiéndolo entre el número de trabajos. El tiempo de sobrecarga, también conocido como latencia, indica cual es el tiempo mínimo que debe mantenerse ocupado con trabajos a un trabajador para que valga la pena asignarle trabajos. La latencia depende de la implementación del maestro, de la carga de procesos del sistema, ancho de banda, uso del disco duro, etc., por lo cual este número no es fijo.

La infraestructura Éxigus pone a disposición la implementación de 3 métodos de tolerancia a fallos (pueden considerarse 4, ya que la no tolerancia a fallos es un caso especial). Existen dos diferencias importantes entre los métodos de tolerancia a fallos: el tiempo y la probabilidad de que los resultados sean correctos. Para el caso del tiempo, se hará referencia a él como el factor que toma un método de tolerancia a fallos respecto a un método sin

tolerancia a fallos. La siguiente tabla resume las fórmulas para determinar el tiempo estimado de término y la probabilidad de correctez de los resultados:

Método de tolerancia a fallos	**Tiempo estimado de término**	**Probabilidad de correctez**
Sin tolerancia a fallos	t	$1-f$
Voto mayoritario	$m*t$	$1-\sum_{j=m}^{2m-1}\binom{2m-1}{j}\varphi^{j}(1-\varphi)^{(2m-1-j)}$
Trabajos descubridores	$t+n*q$	$1-\frac{f}{qn}$
Credibilidad	No existe fórmula	CrW

En donde:

m es el número de votos

t es el tiempo que toma en completarse un proyecto sin usar un método de tolerancia a fallos

φ es la fracción de la población considerada como saboteadora

n es el número de trabajos que conforman un proyecto

q es la tasa de asignación de trabajos descubridores

CrW es el umbral mínimo de credibilidad de para marcar un trabajo como resuelto

Respecto al tiempo que toma credibilidad, el creador de ese método no especificó en sus publicaciones la formula para estimar el tiempo de termino, aunque en base a simulaciones realizadas por él, se estimó que el tiempo de término es menor al de Inspección al azar.

Regresando al proyecto espera, al término del mismo, la probabilidad de correctez para los resultados es:

Sin tolerancia a fallos: 0.8

Voto mayoritario: 0.8844

Trabajos descubridores: 0.9375

Credibilidad: 0.8 y 0.9 (debido a que se ejecutaron dos proyectos con diferentes CrW, con fines comparativos)

Tomando en cuenta los tiempos obtenidos para cada método de tolerancia a fallos y sacando promedios, se obtuvieron los siguientes factores de incremento en el tiempo de término respecto a no tolerancia a fallos:

Voto mayoritario: 2.0012 (2 era el valor esperado)

Trabajos descubridores: 1.1022 (1.10 era el valor esperado)

Credibilidad (CrW = 0.8): 1.1612 (un valor menor a 1.10 era el esperado)

Credibilidad (CrW = 0.9): 2.1090 (igualmente, 1.10 era el límite superior esperado)

En el caso de voto mayoritario y trabajos descubridores se observa como se cumplen los tiempos estimados. Sin embargo, credibilidad representa discrepancias respecto a su valor esperado. Esto se puede deber a los siguientes motivos:

1. No existe fórmula para determinar el tiempo que toma credibilidad, sólo simulaciones.
2. El código para implementar credibilidad es mucho más complejo comparándolo con los otros métodos.
3. La descripción para el algoritmo de credibilidad no contempla el caso en el que todos los resultados repetidos sean iguales, contrario al caso en que existen al menos dos grupos de resultados diferentes.

Queda abierta la posibilidad de probar un proyecto de este tipo bajo condiciones más reales, en donde si se especifica que el 20% de la población es saboteadora, esa población regrese resultados diferentes a los esperados, además de utilizar un número más grande de trabajos. Mientras estas pruebas no se realicen, lo más recomendable es utilizar el método de trabajos descubridores, el cual ofrece tiempos y probabilidades conocidas.

4.2 PROYECTO TRAZADO DE RAYOS

4.2.1 Introducción

El trazado de rayos o raytracing, es uno de los métodos más populares en los gráficos tridimensionales por computadora para obtener la representación bidimensional de un conjunto de figuras tridimensionales y fuentes de luz, denominado escena. Funciona por

medio del trazado del camino tomado por un rayo de luz a través de una escena y el cálculo de la reflexión, refracción o absorción del rayo dondequiera que intersecta un objeto en la misma. Por ejemplo, comenzando de una fuente de luz, se traza un rayo de esa luz hacia una superficie de la escena, la cual es transparente pero refracta el rayo a una dirección diferente, al mismo tiempo que se absorbe parte del espectro (y se altera el color). De aquí, al rayo puede chocar con una superficie que no sea transparente por lo que sufre de absorción (cambiando el color) y reflexión (cambiando la dirección). Finalmente, desde esta segunda superficie el rayo puede ser reflejado directamente en una cámara virtual, en donde su color final contribuye formar la imagen final.

La popularidad del trazado de rayos se debe a su realismo sobre otros métodos de representación. Por ejemplo, las reflexiones y las sombras, que son difíciles de simular en otros algoritmos, surgen de manera natural en el trazado de rayos. La principal desventaja de este método consiste en que puede ser un proceso extremadamente lento, debido a la cantidad de rayos que necesitan ser trazados, y al gran número de cálculos relacionados con la intersección entre los rayos y las superficies geométricas.

Sin embargo, este proceso lento puede acelerarse en cierto modo, si se distribuye la carga entre varios equipos. La distribución de la carga puede hacerse a dos niveles: a nivel de rayo individual o a nivel de escena. Cuando la distribución es a nivel de rayo, cada uno de los píxeles que conforman la imagen final es calculado por un equipo distinto, mientras que a nivel escena, cada una de las mismas es representada por equipos diferentes, logrando un aumento en el desempeño general.

Para la prueba de este proyecto de trazado de rayos, se eligió la distribución de la carga a nivel de escena, debido a que existen una gran variedad de programas perfectamente capaces de representar escenas completas y no pixeles individuales.

4.2.2 Descripción de los trabajos, aplicaciones y herramientas creadas

Aplicación. Se aprovechó una versión vieja del programa de trazado de rayos PovRay. La versión elegida fue la 3.1 (actualmente la versión más reciente es la 3.5) que originalmente fue diseñada para ejecutarse bajo MS-DOS. Se eligió esta versión debido a que era la que ocupaba menos espacio y no requería de un proceso de instalación para ejecutarse (lo único que requiere es un archivo de configuración llamado *povray.ini*). Sin embargo existe el inconveniente de las escenas por representar tienen que ser compatibles con la versión 3.1.

El programa, *povray.exe*, tiene varias opciones que se le pueden indicar por medio de la línea de comandos para representar una escena. Gran parte de estas opciones pueden ser modificadas para obtener distintos resultados. Por ejemplo, se puede obtener una resolución final de imagen de 640 x 480 pixeles usando +w640 +h480 en vez de +w800 +h600. Una descripción detallada de estas opciones se encuentra en la documentación del programa. La línea de comandos usada en este caso en particular fue:

```
povray.exe -d +iEscenaEntrada +oImagenSalida +w800 +h600 +q9
```

donde:

-d es el modificador que indica que no se debe mostrar una ventana con el progreso de la representación

+iEscenaEntrada sirve para indicar la escena que se va a representar. El nombre del archivo se indica a continuación del prefijo "+i"

+oImagenSalida indica el nombre del archivo en que se guardara la imagen generada. El nombre se indica a continuación del prefijo "+o"

+w800 indica que el ancho de la imagen generada es de 800 pixeles

+h600 indica que el alto de la imagen generada es de 600 pixeles

+q9 indica la calidad de la imagen representada

Tomando en cuenta esto, la aplicación está lista para ser distribuida en un archivo denominado *povraydos31.zip* que contiene el archivo ejecutable *povray.exe,* el archivo de inicialización *povray.ini* y la firma digital del contenido, *signature.*

Trabajos. Los trabajos sólo requerían de una escena por representar y la especificación xml del trabajo. El identificador común para varios elementos que conforman el trabajo es el nombre de la escena que se va a representar.

Nombre usado: nombreEscena.zip (en donde el nombre de la escena puede ser cualquiera)

Dependencias: povraydos31.zip

Archivos que contiene: nombreEscena.xml, nombreEscena.pov, signature

Archivos que produce: nombreEscena.tga

Ejemplo de la especificación de trabajo:

```
<?xml version="1.0" standalone="yes" ?>
<Job id='p0009'>
```

```
  <Dependencies>
    <App>povraydos31.zip</App>
  </Dependencies>
  <Tasks>
    <Task id='x' rightExitCode='0' retries='0' timeToRetry='0'>
       <Process>povray.exe -d +ip0009.pov +op0009.tga +w800 +h600 +q9</Process>
    </Task>
  </Tasks>
   <Results>
    <Package fileName='p0009-res.zip' zipLevel='9' upload='http://MASTER/cgi-
                                  bin/resrecv.exe' >
     <FileToPack>p0009.tga</FileToPack>
    </Package>
   </Results>
</Job>
```

Herramientas. Se creó una herramienta, denominada povjbmkr.exe, para crear los trabajos basándose en un conjunto de escenas. El programa toma como parámetro un directorio donde se encuentran los archivos con extensión .pov. Por cada escena encontrada, se crea un trabajo con una clave de trabajo idéntica al nombre del archivo, pero sin la extensión. La escena .pov, junto con la especificación de trabajo son comprimidos en un archivo Zip, que posteriormente se firma, lo cual deja los trabajos listos para ser asignados.

4.2.3 Pruebas realizadas en uno y varios equipos físicos

El hardware y software utilizado son idénticos a los descritos para el proyecto Espera.

Condiciones de ejecución de la infraestructura Éxigus: Se tomaron 21 escenas para PovRay y en base a ellas se creó un número idéntico de trabajos. No se usó ningún método de tolerancia a fallos y se especificó que fuera 1 el número máximo de trabajos por asignar

Resultados:

Las 21 escenas que componen la prueba tardaron en total 116 segundos para ser representadas de forma secuencial sin hacer uso de la infraestructura Éxigus. Usando la infraestructura, los resultados para un número variado de trabajadores fueron:

Un solo equipo

Trabajadores	Tiempo (seg.)
1	132
2	134
4	149
8	165

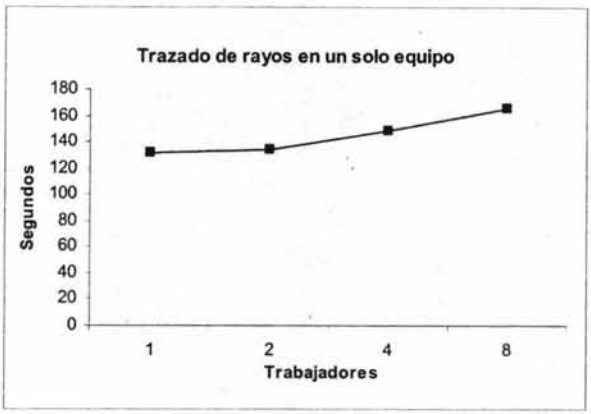

Figura 4.3 Gráfica de resultados para la prueba de trazado de rayos un solo equipo

Varios equipos

Trabajadores	Tiempo (seg.)
1	132
2	97
3	71

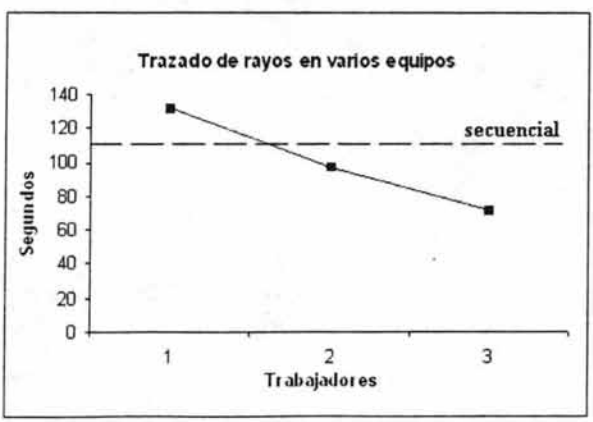

figura 4.4 Gráfica de resultados para la prueba de trazado de rayos en varios equipos

4.2.4 Interpretación de los resultados

Contrario al proyecto anterior, ejecutar más de un trabajador en un solo equipo aumenta el tiempo de término. Esto se debe a que la aplicación PovRay es altamente demandante del procesador, lo cual no deja tiempo libre de procesamiento para una ejecución eficiente de las demás aplicaciones.
Sin embargo, cuando hay un trabajador por equipo físico, se observa una ganancia de tiempo para la conclusión del proyecto. A pesar de la sobrecarga que implica Éxigus por sí mismo, con sólo dos trabajadores se pudo concluir el proyecto en un tiempo menor al necesario a una ejecución secuencial. Más específicamente, para dos equipos físicos el proyecto se ejecutó 1.19 veces más rápido[15] con respecto a la ejecución secuencial sin el uso de la infraestructura. En el caso de 3 equipos la conclusión o aceleración es 1.63 veces más rápida que su contraparte secuencial. Si estos cálculos se realizan contra el tiempo de 1 sólo trabajador usando la infraestructura Éxigus, la aceleración para 2 equipos es de 1.36 y para 3 equipos es 1.85, lo cual es lejano a lo que se esperaría, es decir, el incremento en los equipos debería proveer una aceleración proporcional. Por ejemplo, para dos equipos la aceleración esperada debería ser alrededor de 2, de la misma forma que para tres equipos, la aceleración debería reflejar una reducción de tiempo de alrededor de la tercera parte.

Debido a la orientación de Éxigus, lograr esta clase de aceleraciones no es posible, como se puede constatar. Sin embargo, su valor se encuentra en todas las características agregadas que posee.

4.3 PROYECTO FACTORIZACIÓN

4.3.1 Introducción

Todos los números enteros pueden ser expresados como un producto de números primos, aunque ello no quiere decir que se puedan encontrar fácilmente los números primos para un entero dado. El problema de determinar cuales son los números primos que componen un

---

[15] Este número se obtiene dividiendo el tiempo de un solo trabajador entre el tiempo de varios

número es denominado factorización. Existen varios métodos sencillos para factorizar, pero todos tienen algo en común: el tiempo requerido para factorizar un número se incrementa exponencialmente conforme aumenta el número de dígitos del mismo. Uno de los métodos más simples, el de prueba por división, consiste en dividir el número que se desea factorizar entre todos los posibles números desde 2 hasta la raíz cuadrada del número que se desea factorizar. Este método funciona bien mientras no se trate de factorizar números con más de 13 dígitos.

Existen otros métodos como Pollard p-1, Pollard rho, Williams p+1 y el algoritmo de fracción continuada, pero todos tienen el problema que se vuelven tan lentos como prueba por división conforme el número por factorizar se hace más grande o si el número es especialmente resistente a la factorización por estos métodos.

Los métodos más avanzados para factorización son la criba cuadrática de campo (QFS por las siglas en inglés de Quadratic Field Sieve), la curva elíptica y la criba del cuerpo de números (NFS por siglas en inglés de Number Field Sieve). NFS es actualmente el método de factorización más rápido y se utilizó en 1996 para factorizar un número de 133 dígitos, mientras que en 1999 se usó para la factorización de un número de 155 dígitos. Aunque no existe un algoritmo conocido para factorizar en tiempo polinomial, se sabe que la factorización no es un problema NP completo. De hecho, un algoritmo para factorizar en tiempo lineal puede ser implementado usando una computadora cuántica.

La implementación de cualquiera de estos tres últimos algoritmos representa por si misma un auténtico reto. Es necesario dominar las matemáticas complejas que involucran estos algoritmos. Sin embargo, gracias a la generosidad de sus creadores, existen al menos dos implementaciones disponibles en Internet para el algoritmo QFS, y lo que es aún mejor, están diseñadas para lograr la factorización de un número de forma distribuida. En una de las implementaciones, el equivalente al maestro estaba programado en lenguaje C, mientras que los equivalentes clientes requerían de una máquina virtual de Java para ejecutarse, debido a estar programados en el mismo lenguaje. La otra implementación requería de un sistema capaz de manejar la interfaz de paso de mensajes o MPI (por las siglas en inglés de Message Passing Interface). Como es fácil de darse cuenta, ambas implementaciones tienen requerimientos de software que no se encuentran comúnmente en los equipos de cómputo.

Por lo tanto, si se quería factorizar un número por medio de QFS distribuido, era necesario modificar y adaptar alguna de las implementaciones. Se eligió la implementación basada en MPI debido a que era la que contaba con la mejor documentación.

En la versión original de la factorización por QFS usando MPI[16], un nodo del sistema, denominado Maestro, atiende las solicitudes de nodos denominados esclavos, los cuales realizan la mayor parte del trabajo. Un nodo esclavo le envía un mensaje al nodo Maestro indicando su disponibilidad. Entonces el nodo maestro le envía el número sobre el cual se está realizando la factorización, denominado N, y un elemento de una secuencia de números, denominado D. En base a N y a D el esclavo puede obtener cero o más elementos (denominados relaciones) que forman parte de 3 arreglos distintos, denominados Afx, Axb y matriz de exponentes. Cada relación obtenida tiene uno y sólo un elemento para cada uno de los 3 arreglos distintos. Cuando el esclavo ha terminado de analizar el N y D, le indica al maestro cuantas relaciones obtuvo y consecuentemente le envía estos datos. El maestro recolecta los resultados y sigue enviando N y D a los esclavos que se lo soliciten hasta que se hayan obtenido mínimo R relaciones. El número de relaciones por obtener depende mucho del número a factorizar y no tanto del tamaño del mismo. Por ejemplo, para dos números de 50 dígitos, puede darse el caso que el primero requiera de 535 relaciones, mientras que el otro puede requerir hasta 2350. Cuando el maestro ha juntado las R relaciones que necesita, puede obtener por si mismo y en un tiempo mínimo, la factorización del número N.

Para adaptar esta versión QFS basada en MPI, se modificó el código del nodo cliente para que en vez de enviar y recibir mensajes, leyera los números N y D de dos archivos distintos, y las relaciones encontradas las guardara en tres archivos, uno para los elementos del arreglo Afx, otro para Axb y un último para la matriz de exponentes. El código del nodo maestro se modificó para que generara los trabajos basándose en la secuencia de números D y las relaciones que se necesitaban. Principalmente con estas modificaciones, fue posible realizar la factorización de un número grande usando la infraestructura Éxigus.

---

[16] Consultar http://www.daimi.au.dk/~michael/qs/

4.3.2 Descripción de los trabajos, aplicaciones y herramientas creadas

Aplicación. Para la implementación del algoritmo QFS de forma distribuida, se comenzó primero por crear un aplicación que dado el número N por factorizar y un número D, fuera capaz de encontrar las relaciones posibles y crear un vector con los valores resultantes para la matriz Axb, Afx y la de exponentes. El resultado fue un programa ejecutable, denominado *qfscli.exe*. Este programa lee los siguientes argumentos de la línea de comandos:

```
qfscli.exe archNum archD archAfx archAxb archExpMtx archRels
```

Donde:

archNum es el nombre del archivo en formato ASCII que contiene el número a factorizar.

archD es el nombre del archivo en formato ASCII que contiene una secuencia de números D sobre los cuales trabajar.

archAfx es el nombre del archivo en donde se guardarán los resultados del arreglo Afx para las relaciones encontradas.

archAxb es el nombre del archivo en donde se guardarán los resultados del arreglo Axb para las relaciones encontradas.

archExpMtx es el nombre del archivo en donde se guardarán los resultados de la matriz de exponentes obtenida para las relaciones encontradas.

archRels es el nombre del archivo en donde se indicarán el número total de relaciones encontradas para toda la secuencia de números D proporcionada.

Tomando en cuenta esto, la aplicación está lista para ser distribuida en un archivo denominado *qfscli-1.00.zip* que contiene el archivo ejecutable *qfscli.exe* y la firma digital del contenido, *signature*.

Trabajos. Una vez teniendo la aplicación, la creación de los trabajos fue muy simple. Solo bastaba agregar el número por factorizar, junto con la secuencia de números D, para que la aplicación tuviera los elementos necesarios con los cuales trabajar.

Nombre usado: qfsXXXXX.zip (donde X puede ser cualquier dígito)
Dependencias: qfscli-1.00.zip
Archivos que contiene: qfsXXXXX.xml, num.txt, d.txt, signature
Archivos que produce: afx.txt, axb.txt, expMtx.txt, rels.txt

Descripción del contenido de cada archivo que contiene el trabajo

num.txt	Aquí se especifica el número por factorizar en formato ASCII.
d.txt	En este archivo se encuentran una secuencia de números en formato ASCII, cada uno en una línea de texto diferente.
qfsXXXXX.xml	Es la especificación del trabajo

Ejemplo de la especificación de trabajo:

```
<?xml version="1.0" standalone="yes" ?>
<Job id='qfsXXXXX'>
  <Dependencies>
    <App>qfscli-1.00.zip</App>
  </Dependencies>
  <Tasks>
    <Task id='x' rightExitCode='0' retries='0' timeToRetry='0'>
       <Process>qfscli.exe num.txt d.txt afx.txt axb.txt expMtx.txt rels.txt</Process>
    </Task>
  </Tasks>
   <Results>
    <Package fileName='qfsXXXXX-res.zip' zipLevel='1'
                                         upload='http://MASTER/cgi-bin/resrecv.exe' >
     <FileToPack>afx.txt</FileToPack>
     <FileToPack>axb.txt</FileToPack>
     <FileToPack>expMtx.txt</FileToPack>
     <FileToPack>rels.txt</FileToPack>
    </Package>
   </Results>
</Job>
```

Herramientas. Se crearon tres herramientas adicionales a los trabajos y la aplicación. La primera herramienta, denominada *qfsjbmkr.exe*, tiene la función de crear los trabajos para un número dado. Sin embargo, aquí se presentó un problema muy interesante. Aunque se conoce el número de relaciones que se necesitan para factorizar un número N, usando QFS, se desconoce cuantas relaciones se encontrarán por cada número D sobre el cual se trabaje. En algunos casos un solo número D puede regresar todas las relaciones necesarias mientras que en otros se obtienen en promedio 1 ó 2 relaciones por cada 10 números D. Bajo observaciones empíricas se encontró que la distribución de las relaciones encontradas es

relativamente uniforme. Viendo esto, la herramienta que realiza los trabajos obtiene por sí misma el 1% de las relaciones totales y en base a ello hace un estimado de cuantos números D se deben generar. Debido a que el estimado puede ser inexacto, los trabajos se crean sobre el cálculo estimado de números D más un 70%. Una vez conociendo el total de números D sobre los cuales trabajar, el usuario de la herramienta especifica el número de trabajos que se desean crear y en cada trabajo se agrega la fracción correspondiente de números D, junto con el número por factorizar y la especificación de trabajo.

La otra herramienta, denominada *qfsMonitor*, tiene relación con el desconocimiento de las relaciones que se obtendrán para un número D. Debido a que los trabajos se hacen sobre un estimado, existe la posibilidad de obtener el número total de relaciones necesitadas antes de que todos los trabajos sean resueltos, desperdiciando con ello tiempo. La solución a este problema fue crear un programa que se ejecuta del lado del maestro. Este programa constantemente revisa por los trabajos que el maestro Éxigus haya marcado como resueltos y en base a ellos lleva la contabilidad de las relaciones obtenidas hasta el momento. En el momento en que las relaciones sean mayores o iguales a las necesarias, el programa *qfsMonitor* marca por sí mismo los trabajos restantes como resueltos para que no se asignen más. Una vez hecho esto, junta todas las relaciones y guarda los respectivos datos en tres archivos: *afx.txt*, *axb.txt* y *expMtx.txt*, los cuales se usarán en el paso final de la factorización.

Por último, la tercera herramienta, denominada simplemente *qfs.exe*, toma los archivos con los arreglos Afx, Axb y la matriz de exponentes, junto con el número a factorizar, y trata de obtener la primera factorización no trivial para el número dado. El resultado de la factorización lo guarda en dos archivos, *factorA.txt* y *factorB.txt*, los cuales contienen los dos factores del número dado en una representación ASCII.

4.3.3 Pruebas realizadas en uno y varios equipos físicos

El hardware y software utilizado son idénticos a los descritos para el proyecto Espera

Condiciones de ejecución de la infraestructura Éxigus: Se crearon 50 trabajos. No se usó ningún método de tolerancia a fallos y se especificó que fuera 1 el número máximo de

trabajos por asignar. Del lado del Maestro se ejecutaba al mismo tiempo la herramienta *qfsMonitor* para determinar el momento que debían parar la asignación de trabajos.

Número factorizado: El número elegido para factorizarse fue el 227° número fibonacci, el 123227981463641240980692501505442003148737643593, que tiene 48 dígitos.
La factorización de este número es:
23609 x 5219534137983025159078847113619467285727377, la cual se comprobó usando la infraestructura Éxigus.

Resultados:
El número tardó 89 segundos para ser factorizado de forma secuencial sin hacer uso de la infraestructura Éxigus. Usando la infraestructura, los resultados para un número variado de trabajadores fueron:

Un solo equipo

Trabajadores	Tiempo (seg.)
1	125
2	126
4	126
8	135

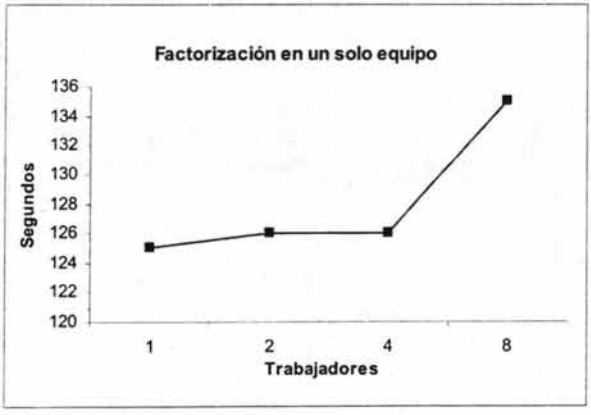

figura 4.5 Gráfica de resultados para la prueba de factorización un solo equipo

Varios equipos

Trabajadores	Tiempo (seg.)
1	125
2	93
3	67

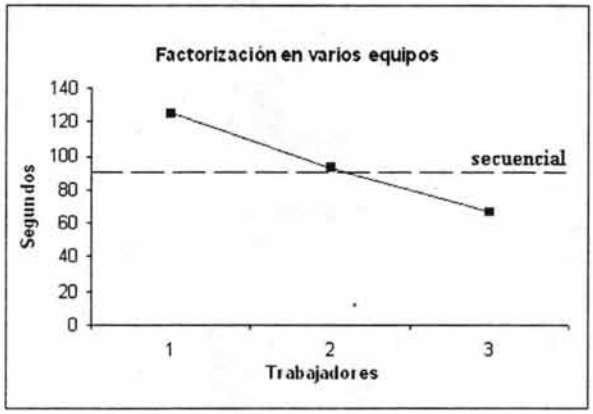

Figura 4.6 Gráfica de resultados para la prueba de factorización en varios equipos

4.3.4 Interpretación de los resultados

Realmente no hay muchos comentarios que realizar sobre los resultados que no se hayan realizado ya. Si se toma la aceleración respecto al tiempo de un solo trabajador, la aceleración para dos y tres equipos es 1.34 y 1.86 respectivamente, lo cual es muy parecido al caso del proyecto de trazado de rayos. Sin embargo, como se puede notar en la gráfica, sólo hasta que se usan 3 trabajadores en equipos físicos distintos es cuando se obtiene una ganancia de tiempo respecto a la ejecución secuencial de la factorización, la cual es de 1.36. Estos datos sugieren que mientras no se usen aplicaciones optimizadas para algún tipo de procesador en particular, se pueden esperar las mismas aceleraciones para un grupo de equipos físicos y diferentes proyectos.

4.4 CONCLUSIONES

En este capítulo se mostraron las pruebas realizadas usando la infraestructura Éxigus. A pesar de ser solamente tres los proyectos probados, proporcionan un panorama de cómo puede ser usada la infraestructura en proyectos variados y los aspectos a tomar en cuenta en cuanto al momento de usarla. Solamente probando la infraestructura en diversos proyectos es como se pueden distinguir las fortalezas y debilidades y por tanto, determinar si es útil a los intereses propios. Respecto a las debilidades, en el siguiente capítulo, se ofrecen una serie de recomendaciones en cuanto a cómo mejorar la infraestructura en varios aspectos.

5. RECOMENDACIONES Y CONCLUSIONES

Una vez que se ha dado por terminado el trabajo del diseño, implementación y prueba de la infraestructura Éxigus, sólo faltan realizar dos cosas: 1) dar una serie de recomendaciones sobre las formas en que se puede llevar a cabo una mejora continua y seguimiento de la infraestructura Éxigus y 2) indicar las conclusiones a las que se llegaron después de dar por concluido el proceso de la realización de esta tesis.

5.1 RECOMENDACIONES

La infraestructura Éxigus presentada, desarrollada, implementada y probada a lo largo de todo este trabajo es funcional, eficiente, portable, segura, mantenible y fácil de usar, en mayor o menor grado, dependiendo de quien, para qué y como se use. Sin embargo hay aspectos que se pueden mejorar o adicionar, los cuales quedan a criterio de quien se decida a hacerlo. La disposición del código fuente y este documento son los medios principales para llevarlo a cabo.

La siguiente es una lista de características, algunas más fáciles de implementar que otras, que se pueden adicionar al trabajo realizado y que de alguna forma, lo mejoran. Los elementos de esta lista no se presentan de acuerdo a algún criterio de ordenación específico,

como podría ser un menor o mayor grado de dificultad, sino más bien quedan a consideración de quien decida hacerlo, de acuerdo con sus conocimientos y experiencia.

1. En el caso de proyectos en los que se pretenda la colaboración o participación abierta sin ánimo de lucro, es recomendable que las estadísticas de cada trabajador se desplieguen en un página Web junto con información de la persona que colabora donando tiempo de PC. Se puede hacer esto más atractivo permitiendo que las personas que colaboran formen grupos y así sumar sus tiempos. Cada determinado periodo se pueden realizar premiaciones simbólicas a aquellos grupos y colaboradores individuales destacados, para así fomentar la competencia entre todos.
2. En la implementación del trabajador se utiliza un programa externos para transferir archivos desde y hacia un servidor Web, tanto en modo seguro como en modo inseguro. "Llamar" éste programa externo involucra la creación de procesos ajenos que afectan en cierta medida el desempeño del trabajador. Aquí la recomendación consiste en integrar la funcionalidad necesaria en el trabajador por medio de bibliotecas estáticas o dinámicas y así evitar crear nuevos procesos continuamente.
3. El trabajador se ejecuta siempre en prioridad normal del sistema, por lo que en algunos casos se puede degradar el desempeño general del equipo donde se ejecuta. La recomendación consiste en permitirle al dueño del equipo determinar con que prioridad desea que se ejecute el trabajador para que no afecte sus intereses.
4. El trabajador utiliza varios archivos en formato XML y texto plano para apoyarse en la realización de sus deberes. El problema con estos archivos es que son fácilmente entendibles, por lo que es recomendable codificarlos (no cifrarlos) de cierta forma que obscurezcan su contenido y evitar alteraciones e inconsistencias casuales y por tanto, posibles colapsos en la ejecución del trabajador.
5. Toda aplicación que se ejecuta en el trabajador tiene que estar preparada para terminar su ejecución en cualquier momento, a petición explicita o causas externas. En caso de contar con el código fuente de la aplicación a ejecutar, simplemente se puede agregar funcionalidad para que se creen puntos de restauración y así continuar las operaciones en caso de terminación externa. En el caso de no contar con el código fuente el problema es mucho mayor, ya que se debe crear copias continuas de la imagen del programa en ejecución, junto con los recursos que esté usando, lo cual por si mismo es un reto importante.

6. Un ambiente de caja de arena es aquel en el cual se ejecuta un programa y limita el accionar del mismo. De esta forma, se pueden evitar los daños ocasionados por virus o código potencialmente dañino. Existen ambientes de caja de arena comerciales o de libre distribución que se pueden usar, pero que ocupan un espacio considerable en disco duro, lo cual afecta el tiempo de descarga y limita la participación de posibles colaboradores. Si no es posible obtener un ambiente de caja de arena pequeño, sería conveniente evaluar la integración de uno grande, tomando en cuenta que cada vez más personas tienen acceso a banda ancha y son más concientes de los riesgos de seguridad.
7. La forma más rápida de generar los trabajos que forman un proyecto es por medio de pequeños programas concretos y guiones de ejecución (archivos de programación de lotes en Windows y Shell en Unix) para escribir archivos en formato XML, comprimir en formato Zip, firmar digitalmente, copiar, cortar, borrar archivos, etc. Sin embargo, esta clase de conocimientos no están disponibles al alcance de todos. Para facilitar la implementación de un proyecto, se pueden crear programas con interfaces gráficas que guíen y faciliten, en la medida de lo posible, a las personas para generar los trabajos que conforman un proyecto. Aquí la dificultad radica en que no existe una forma única de partir un problema grande en otros más pequeños y conformar los trabajos, ya que la mayor parte de los proyectos tienen naturaleza única.
8. El maestro está formado por un servidor Web, un DBMS y un programa que en su concepción más simple, coordina a ambos. Estos tres programas se pueden ejecutar en distintos equipos cada uno. Sería recomendable repetir las pruebas realizadas, para un maestro con un ambiente de ejecución en múltiples equipos y observar la ganancia de desempeño.
9. No todos los sistemas operativos se comportan igual, en especial aquellos de la familia Unix. Siempre ha sido de interés determinar cual es el mejor sistema operativo para una aplicación y una carga de trabajo dada. Sería interesante observar las variaciones de desempeño en la ejecución del Maestro para distintos sistemas operativos. Esto se facilita en gran medida ya que el servidor Web Apache y el DBMS MySQL están disponibles para gran número de sistemas operativos, tanto de Windows como familias de Unix. Ya que el programa que coordina al servidor Web y al DBMS en el Maestro está escrito en Java, también facilita este tipo de pruebas debido a la conocida portabilidad de Java.

10. Los mensajes que se intercambian entre el maestro y los trabajadores son cortos en la implementación actual. Una futura extensión de estos mensajes para contener mayor información implicaría un mayor consumo de ancho de banda, por lo que sería ideal comprimir estos mensajes antes de enviarlos.
11. Así como está definida e implementada la infraestructura actual, el manejo de varios maestros para así atender a más trabajadores está realizado de forma parcial. Una característica muy atractiva sería poder generar dos clases de maestros: uno primario y uno secundario. Un maestro primario se encargaría de atender solamente a maestros secundarios, mientras que estos últimos sería los encargados de atender a los trabajadores.
12. Una vez que se cuenten con varios maestros y una jerarquía entre ellos, lo ideal es balancear la carga de trabajo total del proyecto de forma dinámica entre todos los maestros. El balanceo dinámico es muy importante ya que permite el ingreso de nuevos maestros al proyecto, una vez iniciado éste, aligerando la carga todos los maestros actuales.
13. Los trabajadores por omisión siempre revisan la firma digital de los trabajos y las aplicaciones que reciben. Sin embargo en ambientes donde se tiene control de los equipos donde se ejecutan, una opción interesante sería poder desactivar o activar a voluntad o por medio de mensajes, la verificación de las firmas, con fines de lograr un mejor desempeño.
14. Los resultados que genera un trabajador para un trabajo dado se pueden enviar cifrados o no hacia el maestro. Sin embargo, esto tiene que definirse al momento de crear el trabajo y no puede ser cambiado. En ciertas circunstancias, sería recomendable indicarle a un trabajador la forma en que deben enviarse los resultados, ya sean cifrados o no.

5.2 CONCLUSIONES

El objetivo principal de esta tesis era la creación de herramientas que facilitaran la realización de proyectos de cómputo distribuido basados en Internet, ofreciendo flexibilidad, escalabilidad y seguridad. Este objetivo se cumplió, aunque no fue de una forma ni sencilla

ni rápida. Hubo una serie de problemas que se presentaron y que se tuvieron que resolver para lograr este objetivo. De la solución de estos problemas se obtuvieron una serie de conclusiones, las cuales se presentan a continuación.

Aunque las abstracciones que forman el modelo maestro-trabajador (obtener trabajos, realizar trabajos, enviar resultados, asignar trabajos, recibir resultados) son sencillas de entender, su implementación es de gran complejidad. El aspecto más difícil por resolver es la forma en que se implementarán los trabajos y la forma de hacerlos. Las opciones de implementación para esto abarcan tecnologías de alto nivel como RMI, RPC, CORBA, DCOM o implementaciones más simples como bibliotecas dinámicas, ejecutables nativos, u objetos puros. Ambas clases de implementaciones son usadas tanto en proyectos como herramientas y determinan la naturaleza de los problemas por resolver.

Una vez resuelto la implementación de los trabajos y la forma de realizarlos, las dificultades no terminan ahí. Es necesario tener conocimientos de comunicación en redes, multiprocesamiento, bases de datos, servidores de archivos, criptografía, compresión de datos y sistemas operativos como mínimo para poder empezar a diseñar e implementar los aspectos que conforman una infraestructura.

En vista de todo lo contenido en una infraestructura para la implementación de proyectos de cómputo distribuido, no existe, al menos por el momento, una herramienta que sea fácil de usar y aprender para aquel que carezca de un nivel adecuado de conocimientos respecto al cómputo distribuido y lo que conlleva. Una vez que se tiene ese nivel de conocimientos es entonces cuando se puede discernir entre usar una u otra herramienta. No en balde algunas compañías que realizan esta clase de herramientas basan su fuente de ingresos no en el licenciamiento del software, sino más bien en el soporte y experiencia que pueden ofrecer en la implementación de un proyecto.

Por otro lado, el cómputo distribuido no es la solución definitiva a las carencias de poder computacional. Existen problemas en los que es necesaria una gran velocidad de comunicación entre los nodos que forman la arquitectura, para realizar el trabajo de forma adecuada, como podría ser una supercomputadora o un cluster.

Aunque el problema se preste al uso de una arquitectura distribuida, hay que determinar el valor de los resultados que se obtengan, así como el riesgo que puede implicar confiar en colaboradores externos. Por ejemplo, si se descubriera un algoritmo distribuido para desarrollar una cura contra el VIH, lo más adecuado sería rentar o comprar un cluster o

supercomputadora, en vez de utilizar una infraestructura abierta basada en Internet. Esto se debe a que los resultados comerciales de lograrse el objetivo serían muy altos, así como los riesgos basados en la pérdida, fuga o alteración de la información. Con el uso de un ambiente de mayor control como un cluster o una supercomputadora, se minimizan estos riesgos en gran medida.

REFERENCIAS

[1]Baldoni, Roberto. Fundamentals of Distributed Computing: A practical Tour of Vector Clock Systms[en línea].En: *IEEE Distributed Systems Online.* feb. 2002 <http://dsonline.computer.org/0202/features/bal_print.htm>[Consulta: 14 mar. 2003].

[2] *Berkeley Open Infrastructure for Network Computing (BOINC)*[en línea].<http://boinc.berkeley.edu/>[Consulta: 14 mar. 2003]

[3]Breshears, C. y Fagg G."A Computation Allocation Model for Distributed Computing Under MPI_Connect"[en línea]. En: *SOUTHERN SYMPOSIUM ON COMPUTING.* <http://pax.st.usm.edu/cmi/fscc98_html/present_html/cp.breshears.html>[Consulta: 14 mar. 2003].

[4]Breshears, Clay. *Writing PVM Applications*[en línea]. <http://csep1.phy.ornl.gov/pvm_guide/node3.html>[Consulta: 14 mar. 2003].

[5]Christopher, Thomas W. *Overview of Parallel Computing*[en línea]. <http://www.iit.edu/~tc/overview.html> [Consulta: 14 mar. 2003].

[6] *climateprediction.net* [en línea].< http://climateprediction.net />[Consulta: 14 mar. 2003]

[7]Deitel, H.M., Deitel, P.J. *C++ Cómo programar.* Prentice Hall, 2a edición, 1999.

[8] *distributed computing - a search WebServices definition*[en línea]. <http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci760724,00.html>[Consulta: 14 mar. 2003].

[9] *distributed.net*[en linea].<http://www.distributed.net>[Consulta: 14 mar. 2003].

[10]Elepar. *What P2P Computing Is and Is not: Why Elepar Has no Peer in P2P Computing Grids*[en línea]. <http://www.elepar.com>[Consulta: 14 mar. 2003].

[11] *FIDA*[en línea].< http://fida.chem.washington.edu/>[Consulta: 14 mar. 2003]

[12]Fields, Scott. *Hunting for Wasted Computing Power*[en línea]. <http://www.cs.wisc.edu/condor/doc/WiscIdea.html>[Consulta: 14 mar. 2003].

[13] *fightAIDS@home*[en línea].<http://fightaidsathome.scripps.edu/>[Consulta: 14 mar. 2003]

[14] *Folding @home*[en línea].<http://folding.stanford.edu/>[Consulta: 14 mar. 2003]

[15]Foster Ian y Kesselman C. *Computational Grids*[en línea]. <http://www.globus.org/research/papers/chapter2.pdf>[Consulta: 14 mar. 2003].

[16]Freedman, Alan. *Diccionario de computación.* Mc Graw Hill, 1993.

[17]Hawick, Ken. *High Performance Computing and Communications Glossary*[en linea] <http://www.npac.syr.edu/nse/hpccgloss/hpccgloss.html>[Consulta: 14 mar. 2003].

[18] *Internet-based Distributed Computing Projects* [en línea].<http://www.aspenleaf.com/distributed/distrib-projects.html>[Consulta: 14 mar. 2003]

[19] *Java Remote Method Invocation - Distributed Computing for Java*[en línea].<http://java.sun.com/marketing/colletaral/javarmi.html>[Consulta: 14 mar. 2003]

[20]Kacksuk Peter y Vadja Ferenc. *Network-based Distributed Computing (Metacomputing)*[en línea] <http://www.ercim.org/publication/prosp/NdBC.pdf>[Consulta: 14 mar. 2003].

[21]Kalakota, Ravi. "What is Distributed Processing?"[en línea]. En: *Client/Server Frequently Asked Questions*.17 ago. 1998.Subject: 2.8 <http://isc.faqs.org/faqs/client-server-faq/> [Consulta: 14 mar. 2003].

[22]Maui High Performance Computing Center. *LINDA INTRODUCTION*[en línea] <http://phi.sinica.edu.tw/instruct/workshop/html/linda/linda.html>[Consulta: 14 mar. 2003].

[23]*Mithral Client-Server Software Development Kit*[en línea].<http://www.mithral.com/products/cs-sdk/>[Consulta: 14 mar. 2003]

[24]*MODELS@home*[en línea].< http://www.cmbi.kun.nl/models/>[Consulta: 14 mar. 2003]

[25]*MoneyBee*[en línea].< http://us.moneybee.net/>[Consulta: 14 mar. 2003]

[26]PLATFORM. Will *P2P Computing Survive or Evolve into Distributed Resource Management ?*[en línea]. <http://www.platform.com/PDFs/whitepapers/p2p.pdf>[Consulta: 14 mar. 2003].

[27]*QADPZ*[en línea].< http://qadpz.sourceforge.net/>[Consulta: 14 mar. 2003]

[28]Sarmenta, Luis. *Volunteer Computing*[en línea]. <http://www.cag.lcs.mit.edu/bayanihan/papers/phd/sarmenta-phd-mit2001.pdf>[Consulta: 14 mar. 2003].

[29]*Scripps Research Institute*[en línea].<http://www.scripps.edu/>[Consulta: 14 mar. 2003]

[30]*SETI@home*[en linea]. <http://setiathome.ssl.berkeley.edu/>[Consulta: 14 mar. 2003].

[31]Software Engineer Institute. *Glossary*[en línea].<http://www.sei.cmu.edu/str/indexes/glossary/ [Consulta: 14 mar. 2003].

[32]Tanenbaum. *Sistemas Operativos Distribuidos*. Prentice Hall, 1996

[33]Tanenbaum. *Sistemas Operativos: Diseño e implementación*. Prentice Hall, 1995

[34]Thuan, L. Thai. *Learning DCOM*. O'Reailly, 1999

[35]Vanhelsuwé, Laurence. "Create your own supercomputer with Java"[en línea]. En: *JavaWorld*. 01 ene. 1997.<http://www.javaworld.com/javaworld/jw-01-1997/jw-01-dampp_p.html>[Consulta: 14 mar. 2003]

[36] *Wikipedia, the free Encyclopedia*[en línea] <http://en.wikipedia.org/wiki/>[Consulta: 09 may. 2004]

[37]Woltman. *Mersenne.org Main Page*[en linea].<http://www.mersenne.org/>[Consulta: 14 mar. 2003].

GLOSARIO

Biblioteca.- Conjunto de rutinas de software precompiladas listas para enlazarse con un programa.

LAN.- Siglas en inglés de Local Area Network. Red de computadoras que cubren un área local, como una oficina o un hogar.

UDP.- Siglas en inglés de User Datagram Protocol. Protocolo mínimo orientado a mensajes de la capa de transporte. UDP no garantiza el envío de un mensaje.

XML.- Abreviación en inglés de eXtensible Markup Language. Es una recomendación para creación de lenguajes de marcado de propósito especial y un subconjunto simplificado de SGML, capaz de describir varias clases de datos.

ANEXO A Preparación del entorno de ejecución del Maestro bajo plataforma Windows

Introducción

El siguiente procedimiento describe los pasos a realizar para configurar el entorno de ejecución del maestro Éxigus. Cabe señalar que el maestro Exigus está formado por tres componentes:

1. Servidor Web
2. Manejador de base de datos
3. Coordinador Exigus

El Coordinador Éxigus requiere de al menos 32 MB de RAM y 10 MB de espacio en disco duro. Los requerimientos mínimos del servidor Web y del manejador de base de datos dependen de la versión que se esté utilizando. Para las versiones indicadas en este documento, un equipo con 128 MB de RAM y 50 MB de espacio libre en disco duro serán suficientes. Los procedimientos de instalación y configuración indicados aquí pueden modificarse en algunos puntos, siendo la experiencia propia la que determinará si es conveniente o no hacerlo. Las versiones de software utilizadas en esta guía no son obligatorias, pero sí recomendables.

Software usado

La siguiente es una lista de las aplicaciones utilizadas en esta guía:

- Windows 2000 Professional
- Apache 1.3.27 (paquete: apache_1.3.27-win32-x86-no_src.msi)
- MySQL 3.23.49 (paquete: mysql-max-3.23.49-win.rar)
- J2SDK 1.4.1(paquete: j2sdk-1_4_1-windows-i586.exe)
- PHP 4.3.2 (paquete: php-4.3.2-Win32.zip) [opcional]
- WinRar 3.20 (paquete: wrar320.exe) [opcional]

Instalación del software

Es muy importante seguir el orden mostrado para evitar posibles inconvenientes.

Winrar 3.20

Sólo es necesario en caso de no contar con un programa para manejar archivos en formato Zip o Rar. La instalación con las opciones por omisión es la más recomendada.

MySQL 3.23.49

1.- Se descomprime el archivo `mysql-max-3.23.49-win.rar` en cualquier directorio, por ejemplo `c:\mysql-max-3.23.49-win`

2.- En el directorio donde se descomprimieron los archivos se encuentra uno denominado `SETUP.EXE`, ese es el archivo que hay que ejecutar para iniciar la instalación.

3.- En un punto de la instalación, se pide el directorio destino de los archivos, el directorio por omisión, `c:\mysql`, es el más recomendable.

4.- A continuación se pide el tipo de instalación. Seleccione "Typical"

5.- Una vez concluida la instalación, se configura MySQL para que se ejecute como servicio, esto con el fin de que se inicie junto con el sistema operativo. Para ello, desde el símbolo del sistema de Windows 2000, escriba:

```
c:\> c:\mysql\bin\mysqld-nt --install
```

6.- Desde el símbolo del sistema escriba:

```
c:\> NET START mysql
```

Esto termina la instalación de MySQL.

J2SDK 1.4.1

1.- Se inicia la instalación ejecutando el archivo `j2sdk-1_4_1-windows-i586.exe`

2.- Si la versión de Windows 2000 no tiene instalado Service Pack 2, el instalador mostrará una advertencia recomendando instalarlo antes. Puede hacerse caso omiso de la advertencia y elegir continuar con la instalación.

3.- El directorio por omisión, `c:\j2sdk1.4.1`, es el recomendado para la instalación.

4.- Para ahorrar espacio, se pueden descartar los siguientes componentes de la instalación, sin ningún perjuicio:

- Native Interface Headers
- Demos
- Java Sources

5.- Para las demás opciones de instalación, las presentadas por omisión son las recomendadas.

6.- Hay que configurar unas variables de entorno. En el panel de control, en el item "Sistema" y en la pestaña "Avanzado", hay que hacer clic en el botón con la leyenda "Variables de entorno...". En la sección de variables de sistema, se hace clic en "Nueva..." y se introducen los siguientes datos:

Nombre de variable: `CLASSPATH`
Valor de variable:

```
.;C:\j2sdk1.4.1\jre\lib\ext\dnsns.jar;C:\j2sdk1.4.1\jre\lib\ext\ldapsec
.jar;C:\j2sdk1.4.1\jre\lib\ext\localedata.jar;C:\j2sdk1.4.1\jre\lib\ext
\sunjce_provider.jar
```

Valor de variable debe ser una sola línea, sin espacios. A continuación se presiona "Aceptar" y otra vez "Aceptar".

Esto termina la instalación de J2SDK

Apache 1.3.27

1.- Se inicia la instalación ejecutando el archivo `apache_1.3.27-win32-x86-no_src.msi`

2.- Cuando el programa instalador pida los datos del dominio de la red y el nombre del servidor deben introducirse los correctos, en caso de disponerlos. En caso contrario, no importa el valor

que se introduzca. Se debe seleccionar la opción de ejecutar Apache como servicio y presionar el botón de "Next"

3.- Para ahorrar espacio, se puede seleccionar en el tipo de instalación "Custom" y desmarcar la opción para instalar la documentación. En esta pantalla también se elige el directorio donde se instalará Apache. La ruta recomendada es `c:\Apache`

4.- Por el momento, es todo lo que se hará respecto a Apache. Más adelante se modificaran algunos archivos de configuración, pero eso será después.

PHP 4.3.2

1.- Se descomprime el archivo `php-4.3.2-Win32.zip` al directorio `c:\php`

2.- Se copian los archivos `c:\php\sapi\php4apache.dll` y `c:\php\php4ts.dll` al directorio `c:\apache\apache`

3.- Se copia el archivo `c:\php\php.ini-dist` a `c:\winnt\php.ini` (hay que renombrarlo)

4.- Se abre el archivo de configuración de apache, `c:\Apache\Apache\conf\httpd.conf` y se agregan las siguientes líneas:

```
LoadModule php4_module c:/php/sapi/php4apache.dll
AddModule mod_php4.c
AddType application/x-httpd-php .php
```

Cada línea tiene un lugar específico para agregarse dentro de ese archivo de configuración. Lo más recomendable es utilizar la opción de búsqueda en el editor de texto usado, para encontrar las secciones que tienen los "LoadModule", "AddModule" y "AddType" y agregar la línea respectiva después de la última similar encontrada.

5.- Ahora sólo falta reiniciar el equipo donde se instaló todo el software.

ANEXO B Uso del ambiente de pruebas contenido en el CD

Introducción

En el CD que acompaña esta tesis se encuentran varios archivos relacionados con la herramienta Exigus. En el archivo "exigus.zip" localizado en el directorio raíz del CD se encuentran comprimidos varios directorios que organizan otros archivos. En particular los directorios "pruebas" y "utilidades" contienen lo necesario para realizar pruebas locales tanto del maestro como del trabajador, bajo sistema operativo Windows, y así comprender mejor su funcionamiento. La siguiente guía ilustra los puntos importantes de cómo usar este ambiente de pruebas.

Antes de continuar, es necesario instalar los programas adicionales que requiere el maestro, es decir, el servidor Web, el manejador de base de datos y el ambiente de ejecución de java, junto con el interprete PHP. Para una descripción detallada de cómo instalar y configurar estos programas, consúltese el Anexo A de esta tesis.

Preparación del ambiente básico de pruebas

Descomprímase el archivo "exigus.zip" en la unidad "C" y en el directorio "exigus". Es muy importante que sea esta unidad y directorio debido varios archivos hacen referencia a esta ruta. Una vez descomprimido el archivo se tendrán los siguientes directorios:

```
C:\exigus\codigoFuente\
C:\exigus\otros\
C:\exigus\pruebas\
C:\exigus\utilidades\
```

Ahora, asegúrese que todos los archivos y directorios de `C:\exigus` no tengan el atributo de sólo lectura.
Existen 3 proyectos con sus respectivos trabajos listos para ser probados, referidos en este documento como "espera", "povray" y "factorizacion". Lo más recomendable es revisar primero el proyecto "espera" para después observar el funcionamiento de los demás.

Prueba del primero proyecto, "Espera"

Guía rápida para iniciar el proyecto

Windows 2000:

1.- Si es el primer proyecto que se prueba, ejecútese el archivo de lote de comandos

```
C:\exigus\pruebas\maestro\prepararMaestro-Win2000.bat
```

2.- Ahora ejecútese el archivo el archivo de lote de comandos

```
C:\exigus\pruebas\proyectos\espera\inicializarProyectoEspera-Win2000.bat
```

Windows 98:

1.- Si es el primer proyecto que se prueba, ejecútese el archivo de lote de comandos

```
C:\exigus\pruebas\maestro\prepararMaestro-Win98.bat
```

2.- Ahora ejecútese el archivo el archivo de lote de comandos

```
C:\exigus\pruebas\proyectos\espera\inicializarProyectoEspera-Win98.bat
```

Windows 98 y 2000:

3.- Ejecútese el siguiente archivo de comandos por lotes:

```
C:\exigus\pruebas\maestro\ejecutarMaestro.bat
```

4.- Ahora ejecute el archivo:

```
C:\exigus\pruebas\trabajadores\base1\ExigusWorker.exe
```

Con esto el coordinador Éxigus habrá iniciado, junto con un trabajador. En ambos casos se verá en su respectiva consola el progreso de las actividades que están realizando ambos. Una descripción detallada de estas actividades se muestra más adelante en este documento.

Explicación y objetivo de los pasos anteriores

Para evitar editar el archivo de configuración del servidor apache, se creó una estructura de directorios bajo el directorio `C:\Apache\Apache\htdocs\`. En el subdirectorio `espera\apps` se copió el archivo que contiene la aplicación específica para este proyecto. En el subdirectorio `espera\jobs` se copiaron 32 trabajos listos para ser distribuidos entre los trabajadores. En el subdirectorio `espera\results` se guardarán los resultados que regresen los trabajadores. Ahora cabe preguntarse, ¿cómo sabe el coordinador Exigus en donde se encuentran los trabajos y la aplicación para un proyecto dado? y ¿cómo sabe en que directorio guardar los resultados? La respuesta a ambas preguntas se encuentra en varios archivos de comandos SQL ejecutado en el paso 2 en ambas plataformas.

Este archivo, primero crea las tablas que forman la base de datos *exig2*, usando instrucciones como:

```
CREATE TABLE workers(
      idWorker     char(64) not null, ...
```

Posteriormente se crean las claves de hasta 27 trabajadores, con instrucciones del tipo:

```
INSERT INTO workers VALUES('idWorker001',-1,NOW(),'','','');
INSERT INTO workers VALUES('idWorker002',-1,NOW(),'' ...
```

A continuación se introducen en la base de datos los nombres de archivo y las claves de los trabajos:

```
INSERT INTO jobs VALUES('esp00001', 1, 'esp00001.zip','','');
INSERT INTO jobs VALUES('esp00002', 1, 'esp00002.zip' ...
```

Para cada registro de trabajo existe un campo que indica la clave del perfil de hardware usado para determinar a quien se le puede asignar un trabajo. Pues bien, para la muestra anterior esa clave de trabajo es 1 y el registro correspondiente es el siguiente:

```
INSERT INTO jobProfiles VALUES(1,5,9999,1,10);
```

El perfil de trabajo anterior indica que el trabajador debe tener un procesador mínimo de 5 Mhz, con un máximo de 9999 Mhz, con mínimo 1 MB de RAM y al menos 10MB de disco duro libre. Como se puede notar, el objetivo es que todos los trabajos se puedan asignar a cualquier trabajador.

La última instrucción SQL que se ingresa en la base de datos es la siguiente:

```
INSERT INTO proyectVars VALUES( 0, -- método de tolerancia a fallos
            2, -- votos minimos usado en voto mayoritario
            0.1, -- tasa q de inspección al azar de trabajadores
            0.2, -- porcentaje de trabajadores considerados saboteadores
            0.9, -- umbral de credibilidad para aceptar un trabajo
            1, -- tiempo de espera para un nuevo trabajador
            1, -- tiempo de espera para solicitar trabajo
            1, -- tiempo de espera para aceptar resultados
            1, -- número máximo de trabajos que se asignarán por
               -- trabajador
            'C:\\Apache\\Apache\\htdocs\\espera\\results\\', -- ruta
                 -- absoluta donde se guardaran los resultados
            'http://localhost/espera/jobs/',  -- prefijo URL para
                                              -- descargar trabajos
            'http://localhost/espera/apps/', -- prefijo URL para
                                              -- descargar aplicaciones

            'espera-1.00.zip'); -- lista de aplicaciones que deben tener
                              --los trabajadores para el proyecto actual
```

Todos estos campos se encuentran documentados en el capítulo 3 de la tesis. Cuando inicia el coordinador Exigus, se leen estos datos de la base de datos. En particular, se puede observar donde se indica que se guardarán los archivos de resultados. El caso de los prefijos URL para descarga de aplicaciones y trabajos requiere de más explicación. Cuando a un trabajador se le asigna un trabajo, se le indica la dirección URL completa de donde debe descargarlo. Para formar esta dirección se une el nombre del archivo del trabajo y el prefijo URL para descargar trabajos.
En una instalación por omisión, el servidor Apache toma el directorio `C:\Apache\Apache\htdocs` como su ruta base para buscar los archivos que servirá a los solicitantes. De esta forma, cuando un trabajador solicita el archivo `http://localhost/espera/jobs/esp00001.zip`, el servidor Apache por omisión intenta leerlo siguiendo la ruta `C:\Apache\Apache\htdocs\espera\jobs\esp00001.zip` que es a donde se copiaron los archivos. Esto mismo se repite para las aplicaciones.

Solamente para dejar en claro el papel del servidor Web cabe preguntarse, ¿cómo saben los trabajadores a donde deben de subir los resultados? Eso se responde observando los trabajos. En la especificación del trabajo, se indica el archivo y la dirección URL del archivo CGI que recibirá los resultados. El fragmento en concreto para un trabajo es el siguiente:

```
...
<Results>
      <Package fileName="esp00009-res.zip" zipLevel="1"
                  upload="http://127.0.0.1/cgi-bin/resrecv.exe">
...
```

En los pasos anteriores se copió el archivo `resrecv.exe` al directorio `C:\Apache\Apache\cgi-bin`, que es el directorio por omisión donde el servidor Apache intenta leer los archivos ejecutables CGI. Cuando se llama a ese ejecutable a través de una solicitud http, el servidor Apache pasa el control al programa CGI *resrecv.exe*. Si se cumplen las condiciones de tiempos y tamaño (detalladas en el capítulo 3, en la implementación del maestro) el archivo que envía el trabajador se guarda en el directorio para recepción de resultados indicado en la base de datos, más en concreto, en la tabla *proyectVars*.

Descripción de las actividades del trabajador y el Coordinador Exigus

(Nota.- Por omisión el Coordinador Exigus se ejecuta mostrando la menor información posible. Para corregir esto, modifique el archivo

```
C:\exigus\pruebas\maestro\master.ini
```

para cambiar la opción `minimal_output no` a `minimal_output yes`)

Cuando inicia el trabajador y el Coordinador Exigus (que por simplicidad se llamará maestro a lo largo de este documento) en ambos se muestran consolas de intérprete de comandos donde muestran su actividad, como muestran las siguientes figuras B.1 y B.2

En el momento en que la versión para pruebas del trabajador inicia, se muestra un menú con las posibles acciones que se pueden realizar. Mientras no se seleccione una opción en particular, el trabajador realizará sus funciones normalmente.

Una vez iniciado el trabajador, lo primero que hace es revisar que todas las aplicaciones con las que cuenta no hayan sido alteradas de alguna forma, si alguna ha sido modificada, la elimina. A continuación se determina el estado por iniciar y empieza el ciclo del trabajador.

Suponiendo que el trabajador inicia sin clave de trabajador, aplicaciones y trabajos, las actividades realizadas son las que a continuación se muestran. Primero, el trabajador solicita una clave de trabajador:

```
***[ESTADO_OBTENER_CLAVE_TRABAJADOR]***
Connecting with Master:[Master local] in [localhost:7154]...
```

Si el maestro no se está ejecutando, se muestra un mensaje relativo a ello, se realiza una espera temporal y se vuelve a tratar la última acción:

```
***[ESTADO_OBTENER_CLAVE_TRABAJADOR]***
Connecting with Master:[Master local] in [localhost:7154]...
Master unreachable

***[ESTADO_ESPERA](505 milis)...****
```

figura B.1 Consola de intérprete de comandos para un trabajador

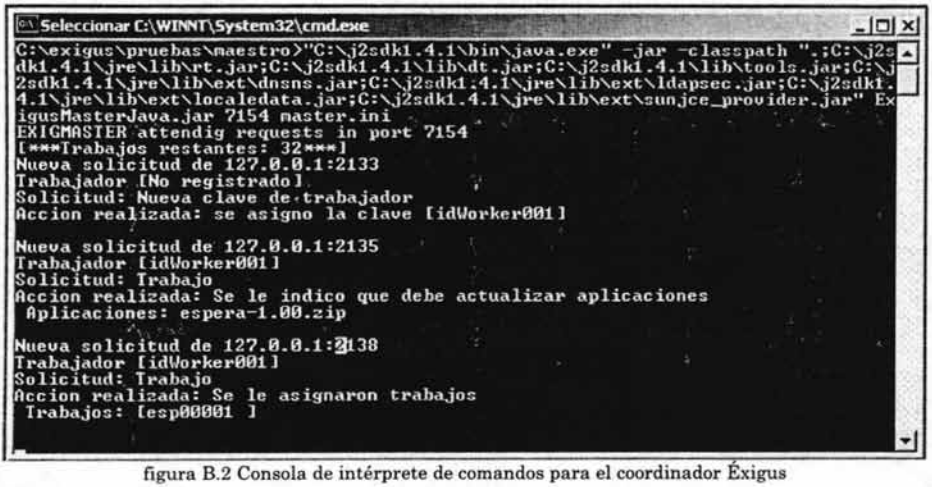

figura B.2 Consola de intérprete de comandos para el coordinador Éxigus

Cuando por fin se puede contactar al maestro, éste revisa si puede asignar una clave de trabajador. En caso afirmativo se muestra un mensaje similar al siguiente en la consola del maestro:

```
Nueva solicitud de 127.0.0.1:2133
Trabajador [No registrado]
Solicitud: Nueva clave de trabajador
Accion realizada: se asigno la clave [idWorker001]
```

Del lado del trabajador se muestra lo siguiente:

```
***[ESTADO_OBTENER_CLAVE_TRABAJADOR]***
Connecting with Master:[Master local] in [localhost:7154]...
Nueva clave de trabajador asignada:[idWorker001]
```

Ahora que el trabajador tiene su clave respectiva, solicita trabajos.

```
***[ESTADO_OBTENIENDO_TRABAJOS]***
Connecting with Master:[Master local] in [localhost:7154]...
```

El maestro recibe la solicitud y determina si el trabajador cuenta con las aplicaciones necesarias para el proyecto actual. Debido a que el trabajador no tiene ninguna, el maestro determina que debe actualizar aplicaciones. En su salida se muestra:

```
Nueva solicitud de 127.0.0.1:2135
Trabajador [idWorker001]
Solicitud: Trabajo
Accion realizada: Se le indico que debe actualizar aplicaciones
 Aplicaciones: espera-1.00.zip
```

El trabajador descarga la aplicación que le indica el maestro. Como no se obtuvieron trabajos, se vuelven a solicitar trabajos:

```
   Descargando archivo [1/1](espera-1.00.zip) del servidor...OK

***[ESTADO_OBTENIENDO_TRABAJOS]***
Connecting with Master:[Master local] in [localhost:7154]...
```

El maestro recibe la nueva solicitud de trabajos. Como ahora sí el trabajador cuenta con la aplicación necesaria, le asigna un trabajo:

```
Nueva solicitud de 127.0.0.1:2138
Trabajador [idWorker001]
Solicitud: Trabajo
Accion realizada: Se le asignaron trabajos
 Trabajos: [esp00001 ]
```

El trabajador descarga el archivo de trabajo y procede a realizarlo:

```
Descargando archivo [1/1](esp00001.zip) del servidor...OK

***[ESTADO_HACIENDO_TRABAJOS]***
Resumiendo trabajo
en:c:\exigus\pruebas\trabajadores\base1\jobs\esp00001\
   Haciendo tarea[1/1]...terminada. Val ret:0
```

Cuando el trabajador termina de hacer todas las tareas para todos los trabajos que tiene por hacer, procede a enviar los resultados:

```
***[ESTADO_ENVIANDO_RESULTADOS]***
Se encontraron 1 archivos para subir
   subiendo archivo[esp00001-res.zip]...OK
```

Del lado del maestro, si el resultado se ha confirmado como válido, lo único que se muestra es que ahora es menor el número de trabajos por hacer:

```
[***Trabajos restantes: 21***]
[***Trabajos restantes: 20***]
```

El trabajador vuelve a solicitar trabajos y el ciclo se repite. Finalmente, cuando el maestro determina que ya no hay más trabajos por asignar, se muestra el tiempo que tomo el proyecto en completarse y le indica a los trabajadores que esperen un momento:

```
[***Trabajos restantes: 0***]

[---Reporte al marcar todos los trabajos como resueltos---]
[]Tiempo del proyecto(ultimo resultado y primera asignacion):68 segs.
[---Fin del reporte---]

Nueva solicitud de 127.0.0.1:2341
Trabajador [idWorker001]
Solicitud: Trabajo
Accion realizada: Se le indica que espere 1 segundo
Motivo:No hay mas trabajos por asignar
```

El trabajador interpreta la respuesta y espera el tiempo indicado por el maestro:

```
***[ESTADO_OBTENIENDO_TRABAJOS]***
Connecting with Master:[Master local] in [localhost:7154]...

***[ESTADO_ESPERA](1000 milis)...****

***[ESTADO_OBTENIENDO_TRABAJOS]***
Connecting with Master:[Master local] in [localhost:7154]...
```

El trabajador continuará solicitando trabajos hasta que el maestro le asigne alguno o hasta que el dueño del equipo donde se ejecuta le indique explícitamente que termine:

```
***[ESTADO_OBTENIENDO_TRABAJOS]***
Connecting with Master:[Master local] in [localhost:7154]...

***[ESTADO_ESPERA](1000 milis)...****
Terminando trabajador...

Se recibio indicacion externa de terminar...
Se recibio indicacion externa de terminar...
*********************************************************
******Terminacion externa. termina ya el programa*****
*********************************************************
```

El texto informativo mostrado no es exclusivo para la versión de pruebas del trabajador. Constantemente el trabajador actualiza el archivo `exigGeneral.xml` para indicar las actividades que está realizando actualmente. Un programa que pueda leer este archivo puede mostrar la información de la forma que le parezca conveniente.
Por ejemplo, la secuencia de figuras B.3, B.4, B.5 y B.6 muestran capturas de imágenes de un protector de pantalla que realiza lo anteriormente descrito. El programa instalador para este protector de pantalla se encuentra en el archivo `C:\exigus\otros\ExigusScrSvr1.00.exe` Del lado izquierdo, en la parte inferior se indica cual es la actividad actual del trabajador.

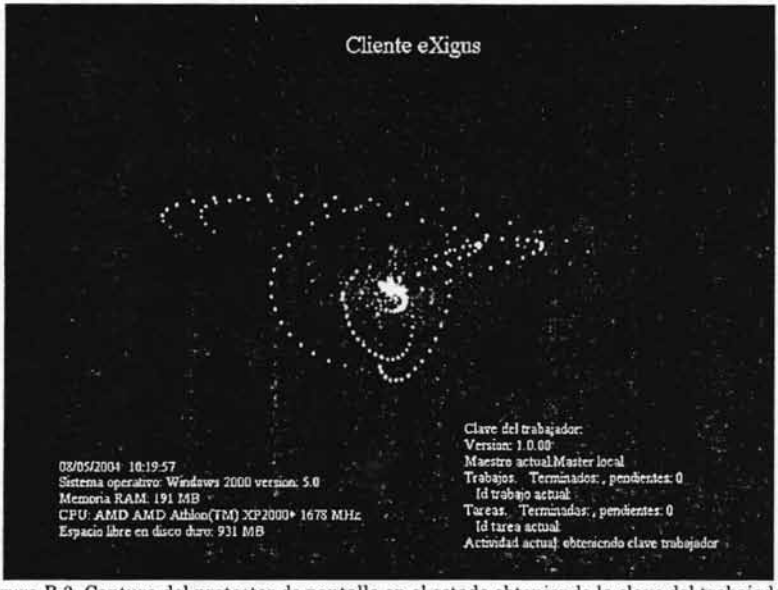

figura B.3 Captura del protector de pantalla en el estado obteniendo la clave del trabajador

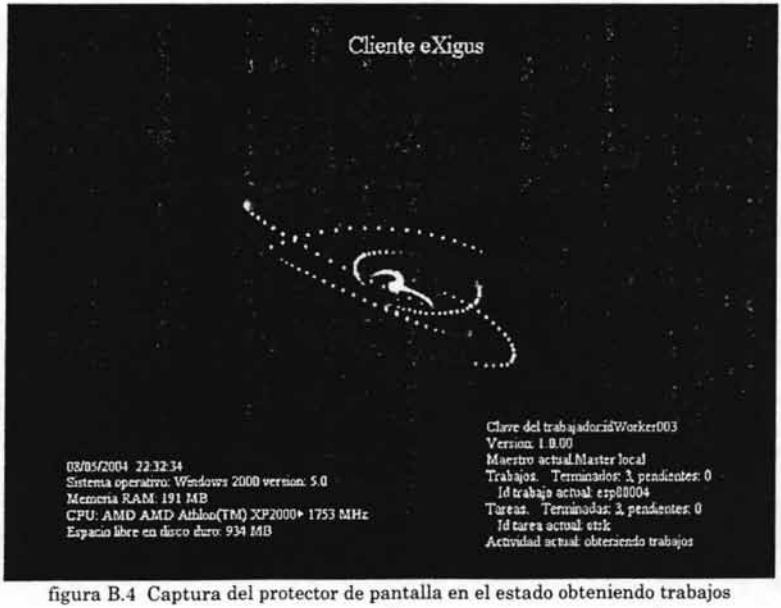

figura B.4 Captura del protector de pantalla en el estado obteniendo trabajos

figura B.5 Captura del protector de pantalla en el estado realizando trabajos

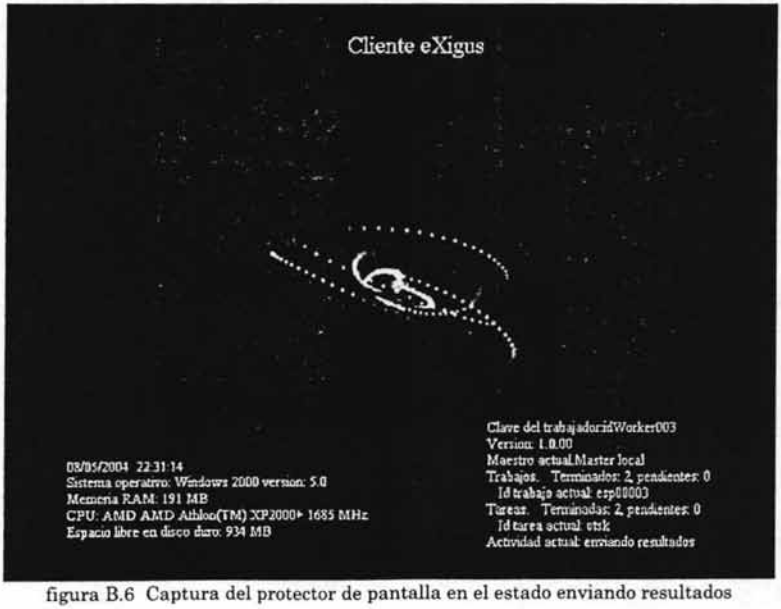

figura B.6 Captura del protector de pantalla en el estado enviando resultados

Modificaciones sobre el proyecto

Numero de trabajadores. La modificación más importante consiste en ejecutar más de un trabajador. Existen hasta 8 trabajadores listos para ejecutarse de forma concurrente. Cabe notar que comparten el mismo CPU, por lo que la ganancia de tiempo para completar un proyecto, podría ser mínima o incluso contraproducente, respecto a una ejecución secuencial. Para ejecutar más de un trabajador, se tiene que invocar el ejecutable correspondiente diferenciado por las siguientes rutas:

```
C:\exigus\pruebas\trabajadores\base1\ExigusWorker.exe
C:\exigus\pruebas\trabajadores\base2\ExigusWorker.exe
C:\exigus\pruebas\trabajadores\base3\ExigusWorker.exe
C:\exigus\pruebas\trabajadores\base4\ExigusWorker.exe
C:\exigus\pruebas\trabajadores\base5\ExigusWorker.exe
C:\exigus\pruebas\trabajadores\base6\ExigusWorker.exe
C:\exigus\pruebas\trabajadores\base7\ExigusWorker.exe
C:\exigus\pruebas\trabajadores\base8\ExigusWorker.exe
```

El número de ejecutables que se invoquen determinará el número de trabajadores de prueba que se tendrán. A pesar de ser el mismo ejecutable, cada uno requiere de una estructura de directorios independiente, de ahí que existan hasta 8 directorios base.

Tolerancia a fallos. Otra modificación interesante consiste en probar cada uno de los métodos de tolerancia a fallos. En el caso de voto mayoritario no se requiere de ninguna modificación especial, salvo por el campo correspondiente en la tabla *proyectVars*. Para el caso de inspección al azar y credibilidad es necesario crear un trabajo inspector. Un trabajo inspector difiere de uno normal con respecto a que se conoce su resultado y por tanto, el SHA1 del mismo, de antemano. En al caso del proyecto espera, se sabe que la aplicación *espera.exe* produce un archivo, *tiempoEsperado.txt* en el que se indican los segundos pasados como parámetro. Debido a que todos los trabajos indican un tiempo de espera de 1 segundo, el SHA1 de un archivo que contiene un solo byte con valor ASCII de "1" , en su representación hexadecimal es "356a192b7913b04c54574d18c28d46e639542
8ab". Este valor se obtuvo creando un archivo denominado "1.txt" con un contenido idéntico al obtenido como resultado de un trabajo. Una vez teniendo este archivo, y haciendo uso del programa OpenSSL.exe, contenido en el directorio c:\exigus\utilidades se escribió el siguiente comando en el intérprete de los mismos:

```
C:\exigus\utilidades>openssl dgst -hex -sha1 < 1.txt
356a192b7913b04c54574d18c28d46e6395428ab
```

lo cual produjo la última línea mostrada. Otra forma de conocer este valor, es leer el campo *result* de la tabla *results* para un trabajo que fue resuelto por un trabajador confiable. Ejemplo:

```
mysql> select idJob, result from results;
+----------+------------------------------------------+
| idJob    | result                                   |
+----------+------------------------------------------+
| esp00003 | 356a192b7913b04c54574d18c28d46e6395428ab |
| esp00004 | 356a192b7913b04c54574d18c28d46e6395428ab |
+----------+------------------------------------------+
```

Conociendo el SHA1 del resultado, es el momento de ingresar un trabajo inspector a la base de datos. Para ello se modificará el trabajo `esp00032.zip` para que tenga esa funcionalidad. En el intérprete de comandos se escribe:

```
C:\mysql\bin>mysql -u root
mysql> USE exig2;
mysql> UPDATE jobs SET flags = 's',result =  '356a192b7913b04c5457
      4d18c28d46e6395428ab' WHERE idJob = 'esp00032';
```

el valor de 's' en el campo *flags* indica que el trabajo es uno de tipo de inspección al azar y el resultado permite comprobar si un trabajador es saboteador o no.

Número de trabajos. El proyecto por omisión sólo contiene 32 trabajos. Si se quiere incrementar o decrementar el número de estos se debe proceder de la siguiente forma. En el directorio `C:\exigus\pruebas\proyectos\espera` existe un ejecutable denominado `waitjbmkr.exe`. Con este ejecutable fueron creados los 32 trabajos contenidos por omisión en el ambiente de pruebas. Para modificar el número de trabajos, se debe editar el archivo `waitjbmkr.cfg`, contenido en el mismo directorio. En particular, hacia el final del archivo se muestra algo similar a lo siguiente:

```
#Numero de trabajos que se crearan
num_jobs 32
```

Ahí se puede modificar el número de trabajos que se generarán. También se puede modificar el directorio donde se guardarán los archivos (por omisión, `C:\exigus\pruebas\proyectos\espera\trabajos\`) y la clave privada con la que son firmados.
Una vez editado el archivo `waitjbmkr.cfg` sólo basta ejecutar `waitjbmkr.exe` para generar los trabajos. Una vez teniendo estos trabajos, deben copiarse al directorio `C:\Apache\Apache\htdocs\espera\jobs`.
Debido a la variación del número de trabajos, debe actualizarse correspondientemente la base de datos. En el mismo directorio donde se crearon los trabajos, se genera un archivo denominado `esperaIns.sql`, el cual contiene las instrucciones SQL para ingresar los trabajos generados en la base de datos. Un ejemplo de las últimas líneas para la generación de 64 trabajos es:

```
...
INSERT INTO jobs VALUES('esp00061', 1, 'esp00061.zip','','');
INSERT INTO jobs VALUES('esp00062', 1, 'esp00062.zip','','');
INSERT INTO jobs VALUES('esp00063', 1, 'esp00063.zip','','');
INSERT INTO jobs VALUES('esp00064', 1, 'esp00064.zip','','');
INSERT INTO jobProfiles VALUES(1,5,9999,1,10);
```

Como las instrucciones generadas son de tipo *INSERT*, es necesario eliminar primero los registros que contenga la tabla *jobs* de la base de datos. Es muy importante que las alteraciones que se hagan en la base de datos se realicen cuando el coordinador Exigus no se este ejecutando, ya que puede ocasionar errores en tiempo de ejecución. Para detener la ejecución del Coordinador Éxigus, sólo hay que cerrar la ventana donde se ejecuta o presionar la combinación *CTRL+C*.

Prueba del segundo proyecto, "Povray"

Realmente no hay mucha diferencia de este proyecto respecto a "espera". Sin embargo, hay que señalar que este proyecto no se puede ejecutar bajo Windows 98.

Guía rápida para iniciar el proyecto

Realícense los siguientes pasos:

1.- Si es el primer proyecto que se prueba, ejecútese el archivo de lote de comandos

```
C:\exigus\pruebas\maestro\prepararMaestro-Win2000.bat
```

2.- Ahora ejecútese el archivo el archivo de lote de comandos

```
C:\exigus\pruebas\proyectos\povray\inicializarProyectoPovRay-Win2000.bat
```

3.- Ejecútese el siguiente archivo de comandos por lotes:

```
C:\exigus\pruebas\maestro\ejecutarMaestro.bat
```

4.- Ahora ejecute el archivo:

```
C:\exigus\pruebas\trabajadores\basel\ExigusWorker.exe
```

Prueba del segundo proyecto, "Factorización"

A diferencia de los proyectos anteriores, los trabajos se generarán al momento. Para ello se hará uso de una interfaz basada en PHP, la cual crea los trabajos necesarios para factorizar un número que puede ingresar un usuario.

Guía para iniciar el proyecto

Asegúrese primero que no se esté ejecutando el Coordinador Exigus. Ahora realícense los siguientes pasos:

Windows 2000:

1.- Si es el primer proyecto que se prueba, ejecútese el archivo de lote de comandos

```
C:\exigus\pruebas\maestro\prepararMaestro-Win2000.bat
```

2.- Ahora ejecútese el archivo el archivo de lote de comandos

```
C:\exigus\pruebas\proyectos\qfs\inicializarQFS-Win2000.bat
```

Windows 98:

1.- Si es el primer proyecto que se prueba, ejecútese el archivo de lote de comandos

```
C:\exigus\pruebas\maestro\prepararMaestro-Win98.bat
```

2.- Ahora ejecútese el archivo el archivo de lote de comandos

```
C:\exigus\pruebas\proyectos\qfs\inicializarQFS-Win98.bat
```

Windows 98 y 2000:

3.- Ejecútense los siguientes archivos de comandos por lotes:

```
C:\exigus\pruebas\maestro\ejecutarMaestro.bat
C:\exigus\pruebas\maestro\ejecutarQFSMonitor.bat
```

4.- Ahora, si no hay ningún trabajador ejecutándose, puede iniciar uno ejecutando

```
C:\exigus\pruebas\trabajadores\base1\ExigusWorker.exe
```

10.- Inicie su navegador de Internet favorito y escriba la siguiente dirección URL:

http://localhost/qfs/fac1.php

Con ello se muestra una página donde puede ingresar el número que se desea factorizar. Una vez definido este número, sólo debe presionar el botón 'Continuar' que se muestra en el formulario presentado.

11.- Los trabajos se crearán automáticamente dependiendo del número introducido y serán asignados a los trabajadores solicitantes. Al final, se muestra una página en su navegador indicando la factorización del número correspondiente.

Descripción de las actividades de las aplicaciones auxiliares del proyecto

Los trabajadores y el Coordinador Exigus se comportan normalmente. Sin embargo, para lograr este uso interactivo de la infraestructura se tuvieron que crear varias aplicaciones y guiones de ejecución complementarios de la infraestructura, los cuales son:

qfsstats.exe lee un archivo, **num.txt**, conteniendo un número expresado en ASCII. Este programa determina si es posible o no factorizar el número especificado. En caso de que no sea posible factorizar el número (debido a que es muy corto en cuanto a número de dígitos o que es un primo probable) se genera el archivo **error.txt**, donde se indica la causa. En caso contrario genera el archivo **stats.tx**, donde se indican mensajes sobre análisis realizados sobre el número.

qfsjbmkr.exe lee un archivo, **num.txt**, conteniendo un número expresado en ASCII. En base a este número y a su archivo de configuración, **qfsjbmkr.cfg**, crea los trabajos necesarios para factorizar el número indicado, junto con dos archivos: uno denominado **minimalRelations.txt**, en el que se indican las relaciones mínimas por obtener para factorizar el número en concreto y el otro llamado **jbmkrprogress.txt** en el cual se muestra el progreso de la creación de los trabajos. Consúltese el capítulo 4, en el proyecto factorización, para saber cómo se generan exactamente los trabajos.

qfsResult.exe lee los archivos **afx.txt**, **axb.txt**, **expMtx.txt** y **num.txt**, conteniendo los arreglos con las relaciones necesarias (vea el capítulo 4). En base a esos archivos, genera otros dos: **factorA.txt** y **factorB.txt**, los cuales contienen la representación ASCII de los dos factores que componen un número.

QFSMonitorInteractive.jar es el programa hecho en Java que se ejecuta en el archivo por lotes `correrQFSMonitor.bat`. Este programa tiene dos funciones. En su primera función, monitorea constantemente los resultados recibidos y cuando se han obtenido suficientes relaciones, genera los archivos **afx.txt**, **axb.txt**, **expMtx.txt** conteniendo los arreglos con las relaciones necesarias (vea el capítulo 4). Su segunda función es llamar a los programas **qfsstats.exe**, **qfsjbmkr.exe** y **qfsResult.exe** cada vez que se indica una nueva solicitud de factorización por medio de los guiones PHP. Cuando se detecta la existencia de un archivo nombrado **newFactorRequest.txt** en el directorio `C:\Apache\Apache\htdocs\qfs`, sabe que es el momento de iniciar una nueva factorización, por lo que llama a los programas correspondientes.

fac1.php le muestra un formulario al usuario para que introduzca un número a factorizar. Cuando se envía el formulario al servidor, se llama al guión **fac2.php**

fac2.php escribe en un archivo en formato ASCII el número introducido en el formulario anterior. Este archivo se llama num.txt y a continuación se procede a llamar al guión **fac3.php**

fac3.php muestra diversos mensajes informativos de acuerdo a las actividades relativas a la factorización. Estas actividades son analizar el número, crear los trabajos, obtener las relaciones y obtener los factores del número especificado.

Las figuras B.7, B.8, B.9 y B.10 muestran como luce desde una página de Web el proceso de la factorización para un número dado.

Por último, sólo hay que señalar que si se desea factorizar un nuevo número, se deben detener primero el Coordinador Éxigus y QFSMonitorInteractive y volverlos a iniciar ejecutando:

```
C:\exigus\pruebas\maestro\ejecutarMaestro.bat
C:\exigus\pruebas\maestro\ejecutarQFSMonitor.bat
```

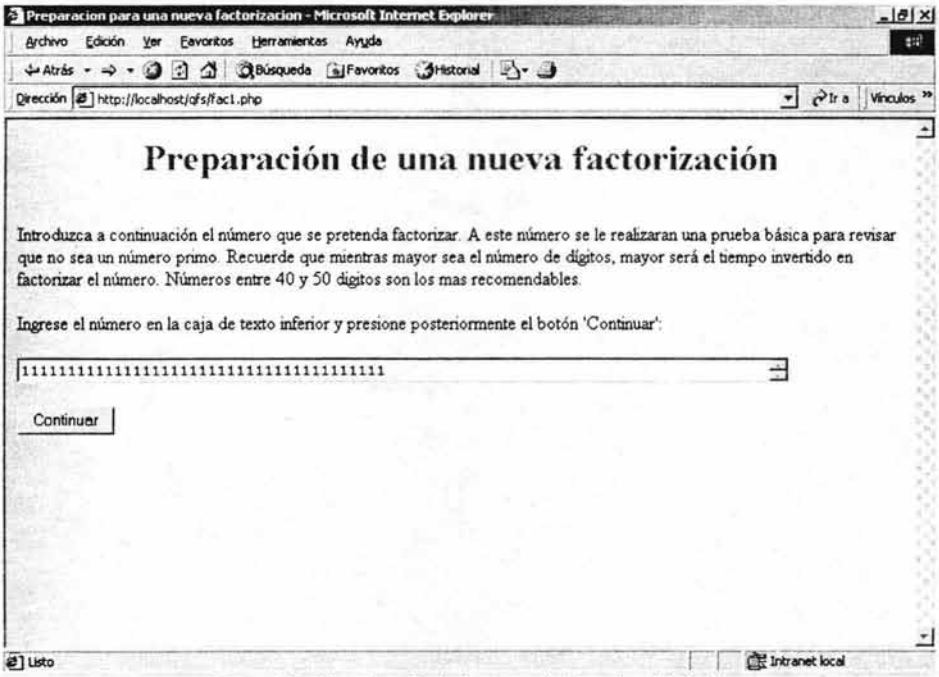

figura B.7 Visualización del proceso de factorización (1/4)

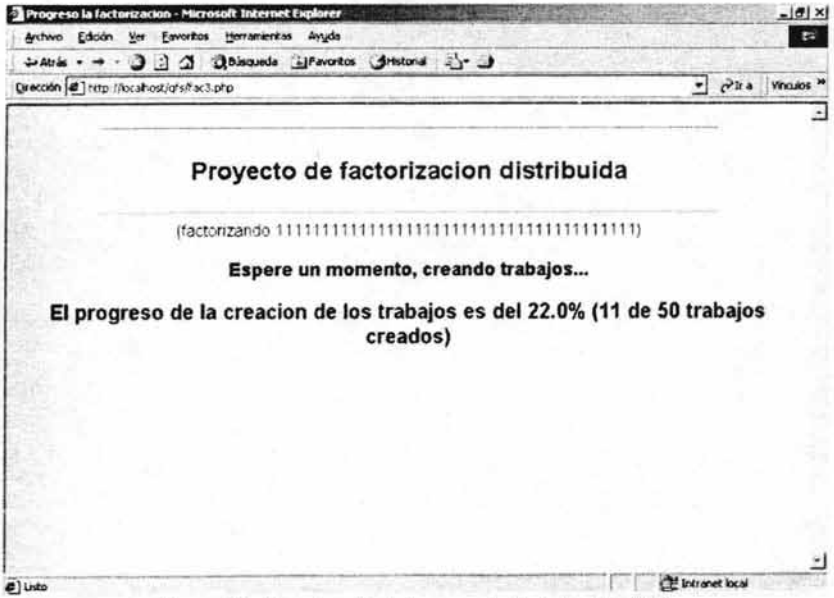

figura B.8 Visualización del proceso de factorización (2/4)

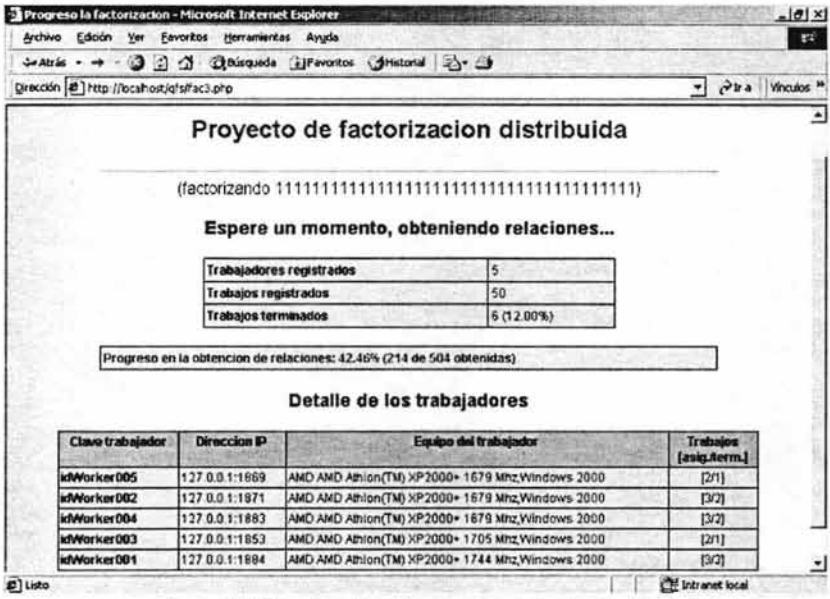

figura B.9 Visualización del proceso de factorización (3/4)

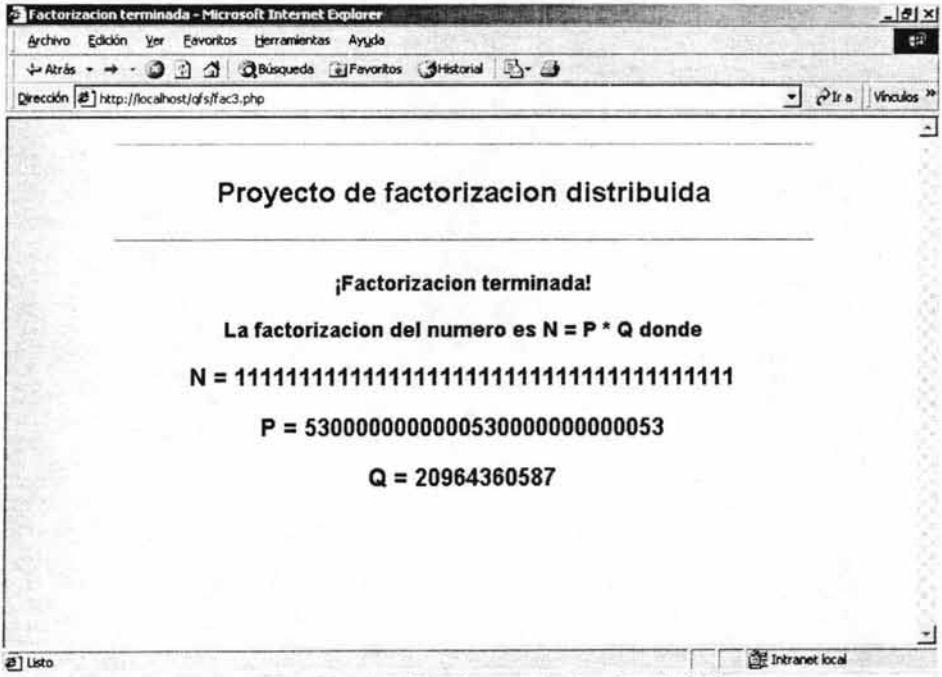

figura B.8 Visualización del proceso de factorización (4/4)

ANEXO C. Guía práctica para el uso de la infraestructura Éxigus

Introducción

La siguiente guía muestra los pasos por realizar para el uso de la infraestructura Éxigus en un proyecto. Con el fin de hacer esta guía lo más útil posible se asume que se tienen conocimientos básicos en la composición y funcionamiento de la infraestructura en su conjunto.

Pasos a seguir

1.- Se debe crear una pareja de clave pública/privada en formato PEM. Para ello se debe usar el programa OpenSSL, disponible en Internet y también en el ambiente de pruebas contenido en el CD, en la subcarpeta *utilidades* del archivo *exigus.zip*. Las instrucciones que se deben escribir a través del intérprete de comandos son:

```
$openssl genrsa -out privKey 1024
$openssl rsa -pubout -in privKey -out pubKey
```

en donde `privKey` y `pubKey` son los nombres de las claves privada y pública respectivamente.

2.- Los trabajos que se generen deben ser firmados digitalmente. Para ello se puede usar el programa `zipSign.exe` disponible en el CD, en la subcarpeta *utilidades* del archivo *exigus.zip*. Desde la línea de comandos se debe escribir:

```
X:\> zipSign.exe trabajo.zip privKey
```

en donde `trabajo.zip` es el nombre del archivo de trabajo y `privKey` es la clave privada con la que se firmará el trabajo en formato Zip. El programa `zipSign.exe` lo que hace es generar un compendio de tipo SHA1 de todos los archivos ordenados alfabéticamente, contenidos en el trabajo. A este compendio se le obtiene nuevamente su compendio y se cifra con la clave privada `privKey` para obtener un archivo denominado `signature`, el cual se agrega al Zip original.

3.- La o las aplicaciones que utilizaran los trabajadores también deben firmarse digitalmente. Para ello se debe proceder como el punto anterior.

4.- Se deben instalar y configurar los programas que forman parte del maestro Éxigus.

5.- Los trabajos y aplicaciones deben copiarse a un directorio al cual el servidor Apache pueda tener acceso. Se debe tomar nota de cual es el prefijo URL para accederlos por medio de un cliente Web.

6.- Se debe crear la base de datos junto con sus tablas. Las instrucciones SQL para realizarlo se encuentran en el archivo *pruebas\maestro\crearBDExigus.sql,* dentro del archivo exigus.zip contenido en el CD. Si se descomprime ese archivo en la c:\, el comando para la creación de la BD es:

```
c:\> mysql -u root < c:\crearBDExigus.sql
```

7.- Se deben crear los registros que identificarán a los trabajadores en la tabla *workers.*

8.- Todos los trabajos a distribuir se deben registrar en la base de datos, en la tabla *jobs.*
Debido a que un trabajo requiere de un perfil de hardware, también se debe crear al menos un registro en la tabla *jobProfiles.*,

9.- En la tabla *proyectVars* se debe crear un registro en el que se indique las opciones para el proyecto actual.

10.- Se debe indicar en el archivo de configuración del coordinador Exigus, `master.ini`, la localización de clave privada `privKey`. Con ello el coordinador Exigus se encuentra listo y puede iniciarse en cualquier momento.

11.- Se debe integrar la funcionalidad del trabajador Exigus a un programa "cliente". Puede ser de cualquier tipo siempre y cuando se mantengan los archivos y directorios necesarios. También es necesario que al momento de que inicia el trabajador Exigus, la clave de registro, *HKEY_LOCAL_MACHINE\SOFTWARE\Exigus2\basePath,* exista y contenga la ruta absoluta hacia el directorio base que requiere el trabajador.

12.- Distribúyase o hágase disponible el programa "cliente" al mayor número de equipos posibles y con ello el proyecto se encuentra listo.

/.-/-./.-/../.-./.-/--//.-/--./../--/.-//../--//.-/.-./.-/.--./

/./-../../...-/.-../---//./-//---/-./
/.../.-/--//---/.-..//./.---/./-..//./..-/--.-//---/-./../.../
/.-../.-/-./../..-.//.-../.-//./-/-./.-/-/.-./---/.--./--/../

/---/-.-./--../ ./-../.-/.-./--./.-//./.-..//.../---/../-..//.-/
/.-/.-..//.-/.-././../-..//./--//./..-/--.-/
/-../.-/-../../-./..-/-/.-./---/.--./---/
/-../.-/-/.../../--/.-//.-/-./..-//.-././-././-//./-../
/---/--./../-/-./---/-.-.//.-/.../---/--/.-././....//-./.-/-/

/.-./---/.--.//---/-.-./--../ ./-../.-/.-./--./.-//./-//../-//.-/
/./..-/--.-//---/--.--/../.-./.-/-.-.//.-.././/---/-../---/-/
/.-./---/.--.//-.--//---/-../.-/-..//--../.-/....//./--/
/---/-.--/---/.--./.-//..-/-//---/-../---/-/
/---/.../---/--/.-././....//..-/-//---/.-./../--/-../.-/
/./-..//---/-././.-../.-..//-./---/--../.-/.-./---/-.-./
/.-/--../ ./.-../-.../---/-./
/-.--//-../.-/-../-./---/-.../
/-../.-/-../../.../---/.-././-././--./

/./.-./.-/.--././-..//./..-/--.-//---/.-..//./...//---/-./
/---/.-././.--.//---/.-./..-/-/..-/..-.//.-../ ./
/.../.-/.--./..-/-.-./---//..-/-//.-././...//../--//-././
/-.--/..-/--//.-./.-/--./..-/.-..//-./..-/
/./..-/--.-//.-../.-/../-.-././.--./.../ ./
/.-/.-./-../---/.--.//.../.-/--//./../-../.-/-./
/.-./.-/-././.-../.-../

/---/--/../.../../..../-.-./..-/--//---/.-././../..-/--.-//./-/

/.../.-/../-.-./.-/.-./--./

/-../.--./
/.-/.-..//./-..//.-./---/.---/./--//---/.-../
/.-/-./..-//---/--./-././-//./..-/--.-//..././/.-/-../../...-/
/.-/-/.../..-/--.//./.-..//./..-/--.-//.-/--./../--/.-/
/---/-../-././-.--/./.-..//./.../.-./../-/.-././...-/../-../
/..././...-/./.-./.-../.-//.../././.---/.-/.../-././--/