Factorización de Números Enteros Grandes Usando General Number Field Sieve en Paralelo

Juan Camilo Corena Bossa

Iván Mauricio Rey Salazar

Asesor

Milton Quiroga Becerra

Departamento de Ingeniería de Sistemas y Computación

Facultad de Ingeniería

Universidad de los Andes

Bogotá D.C., 2006

Tabla de Contenido

Indice de Ilustraciones

Indice de Ilustraciones

Introducción

La seguridad de la criptografía de los sistemas de llave pública se basa en la intratabilidad del problema de factorización de números. En particular las seguridad de las llaves RSA se puede reducir al problema de factorización de números semiprimos. Actualmente el mejor algoritmo conocido para factorizar números grandes es el General Number Field Sieve (GNFS).

Debido a la gran importancia de la seguridad en criptografía con el algoritmo RSA, RSA Laboratories ha creado el concurso RSA Factoring Challenge, para capturar la atención de la academia. El último número factorizado del RSA Factoring Challenge es el RSA-640, con un premio de $20.000 USD. Fue factorizado por F. Bahr, M. Boehm, J. Franke y T. Kleinjung en la Universidad de Bonn [1]. El siguiente número del concurso es el RSA-704 con un premio de $30.000 USD. Actualmente la marca está en el RSA-200 factorizado por las mismas personas [2].

Con el apoyo de la academia es posible conocer el estado actual de las capacidades tecnológicas de sistemas que permitan descifrar llaves RSA. Apoyándose en las marcas actuales sus respectivas dificultades actuales es posible definir nuevos estándares que aseguren una mayor confianza práctica en la seguridad de las llaves.

Nuestro interés particular es resolver el problema de factorizar números grandes en paralelo usando las capacidades computacionales del Centro de Computación Avanzada en Ingeniería MOX. Utilizando tecnologías de computación distribuida en Grid apoyadas en la infraestructura BOINC se realizó la criba necesaria para el GNFS para luego ser procesada en un servidor con una grande cantidad de memoria disponible.

Objetivos

Objetivos Generales

- Realizar una implementación paralela de un algoritmo de factorización
- Instalar una infraestructura de computación distribuida en las salas del Centro de Computación Avanzada en Ingeniería MOX
- Aprovechar los recursos computacionales de las salas del MOX
- Factorizar un número RSA relativamente difícil
- Probar la capacidad computacional de las salas de la universidad
- Comunicar a la comunidad de la existencia de la infraestructura de computación distribuida

Objetivo Específicos

- Paralelizar el algoritmo de factorización de General Number Field Sieve
- Implementar el algoritmo paralelo en una infraestructura de Grid
- Factorizar RSA-100 y RSA110 no paralelos
- Factorizar RSA-110 en paralelo
- Crear las bases para factorizar RSA-704 del RSA Challenge
- Personalizar la infraestructura para publicitar el Grid-Mox-Uniandes
- Crear un protector de pantalla que informe a los usuarios del Grid
- Aprovechar los recursos del Grid en la noche
- Aprovechar los recursos del Grid durante el día mientras trabajan los estudiantes
- Generar estadísticas que sirvan de apoyo al Grid para nuevos proyectos

Marco Teórico

Por que es importante factorizar números grandes.

Los algoritmos criptográficos en general, basan su fortaleza en un problema computacionalmente difícil de resolver, varios de los algoritmos mas empleados en criptografía de llave pública, tienen como problema a resolver la descomposición en factores primos de números grandes. El mas usado en la actualidad es el algoritmo RSA.

El algoritmo RSA

El algoritmo es producto del trabajo de Ron Rivest, Adi Shamir y Len Adleman, de allí que el nombre del algoritmo provenga de sus iniciales, a continuación se dará una explicación del algoritmo basada en [3], situaciones en las que se usa y la importancia de la factorización de números grandes para atacarlo.

Generación de llaves

1. Se eligen dos números primos grandes p y q, según Burt Kaliski de RSALabs, el tamaño de estos enteros debe ser de 1024 bits como mínimo.
2. Calcular $n = p \cdot q$.
3. Calcular $\phi = (n-1)\cdot(q-1)$.
4. Elegir un número e que cumpla las siguientes propiedades $\{e \in \mathbb{N} | e < \phi \wedge gcd(e, \phi) = 1\}$.
5. Encontrar un número d que cumpla la siguiente propiedades $e \cdot d \equiv 1 \bmod \phi$.
6. Selecciones como su llave pública $\{e, n\}$ y como su llave privada $\{d, n\}$.

Cifrado y Descifrado

Para poder cifrar un mensaje empleando el algoritmo RSA, el mensaje debe convertirse en un número M donde $M < n$ y luego se realiza lo siguiente $C = M^e \bmod n$, donde c es el texto cifrado.

Para descifrar C, se hace lo siguiente $M = C^d \bmod n$, y luego se convierte M al mensaje original.

Usos del algoritmo RSA

RSA es por mucho el algoritmo de llave pública mas usado en el mundo actualmente, tanto para cifrado y descifrado como para firmas digitales. En lo que a cifrado se refiere, este algoritmo solo es usado para intercambio de llaves debido a que las divisiones requeridas para cifrar y descifrar son bastante lentas en comparación con otros algoritmos de cifrado.

En particular un uso bastante importante de este algoritmo son los certificados digitales, empleados por los sitios web que desean tener comunicación segura con sus usuarios. A continuación se encuentra el certificado digital del sitio de transacciones del Banco Davivienda, el cual como se aprecia en la figura, usa como algoritmo de clave pública para el sitio web el

algoritmo de cifrado RSA.

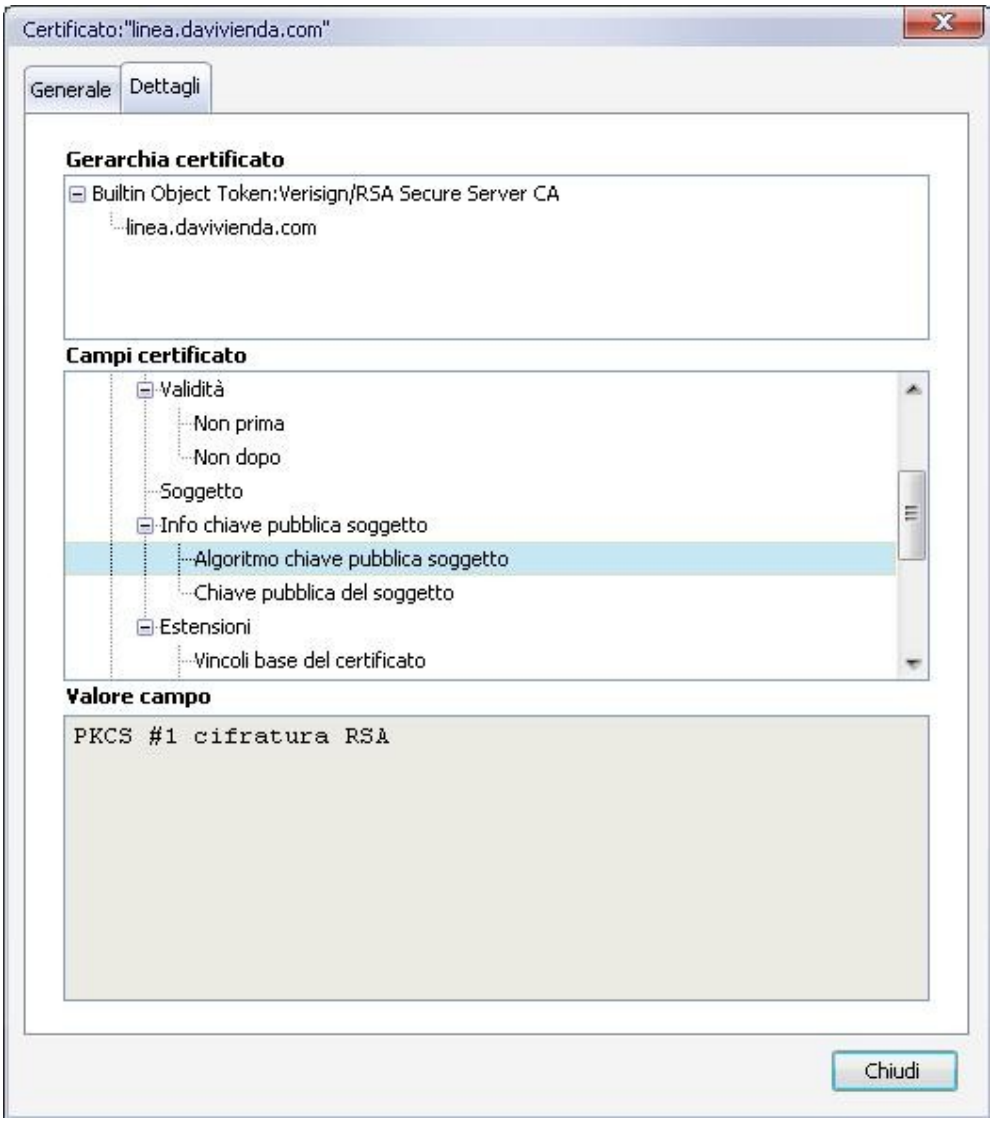


Ilustración 1: Certificado digital del portal de transacciones del Banco Davivienda https://linea.davivienda.com/davivienda.jsp

El certificado también contiene la llave RSA pública del sitio en hexadecimal, por razones de legibilidad esta llave fue convertida a base 10 y dio el siguiente número.

$$n = 269867516625442335983016702384812673432404497906410298350399567398174717422727010929314667635565654403680778712146911511322645716857542756492283776505034360546757058395713830790383482527829306748496310489944235259159205389360452555537776679051906296214188991455071388669826231569115540624014926369601171015022235886796747453982938713292 9$$

La razón por la que solo es un número y no dos como se dijo en la explicación del algoritmo es que en la practica el algoritmo RSA toma como valor de *e* el número 65537 [4].

Ataques sobre RSA

El algoritmo RSA es seguro mientras la llave privada se mantenga conocida solo por su dueño. Como se pudo apreciar en la descripción del algoritmo, las llaves pública y privada se encuentran relacionadas matemáticamente de tal manera que a partir de la llave pública sea computacionalmente imposible generar la llave privada.

Recordemos un poco el algoritmo

- La llave pública es $\{e, n\}$.
- La llave privada es $\{d, n\}$.

Tenemos que e y n aparecen en la llave pública, por lo tanto lo único que se necesita para obtener la llave privada es d, como $(e \cdot d \equiv 1 \ mod \ \phi) = (e \cdot d \equiv 1 \ mod \ (p-1) \cdot (q-1))$, lo que se necesita para poder despejar d, son p y q los cuales son los factores primos de n. Entonces el ataque al algoritmo RSA se reduce a factorizar n.

La fortaleza de este algoritmo radica en la dificultad de factorizar números grandes; el cual es un problema que se ha estudiado mucho en teoría de números, pero que hasta el momento no tiene una solución definitiva. Según [5] "no es claro a que clase de clase de complejidad pertenece el problema de factorizar números. Su problema de decisión equivalente ¿Tiene N un factor menor que M? Se encuentra en las clases de complejidad NP y co-NP (NP complemento). Esto es porque tanto las respuestas de Si o No pueden ser comprobadas si se dan los factores primos junto con sus certificados de primalidad. Se sabe también que este problema se encuentra en la clase BQP (la clase Bounded error, Quantum, Polynomial time) gracias al algoritmo de Shor". Este fragmento de articulo, nos da una idea de lo complicado que es clasificar el problema y de lo difícil que es resolverlo en una maquina de Turing determinista; sin embargo, en un computador cuántico, algoritmos como el de Shor pueden resolver el problema de manera mucho mas eficiente.

Retomando el ejemplo del portal de transacciones del Banco Davivienda si lográramos factorizar n, y realizar un ataque de "Man in the middle" entre el portal y sus usuarios, podríamos obtener las contraseñas de todos las personas que realicen transacciones durante la duración del ataque. En términos generales, además de revelar la información privada de las personas y/o empresas, se podría suplantar la identidad de cualquier usuario de firmas digitales al factorizar n .

Algoritmo General Number Field Sieve (GNFS)

Este es el algoritmo con mejor comportamiento asintótico para factorizar números más grandes que 2^{500} [6], su complejidad esta dada por [7]

$$O\left(e^{(c+O(1))\cdot \ln(n)^{\frac{1}{3}} \cdot \ln(\ln(n))^{\frac{2}{3}}}\right)$$

El algoritmo es bastante complejo, debido a que emplea conceptos matemáticos que están fuera de nuestro dominio y que no son el tema central de este trabajo, sin embargo intentaremos explicar su funcionamiento como sustento teórico de las actividades realizadas durante el semestre.

El algoritmo GNFS es un algoritmo que emplea cribas para realizar la factorización de números, según [7] la idea consiste en encontrar dos números

$$\{x, y \mid x^2 \equiv y^2 \ mod \ n \wedge x \neq \pm y \ mod \ n\}$$

porque esto implica que $gcd(x-y, n)$ es un factor no trivial de n.

Para realizar este proceso el algoritmo GNFS se divide en 6 partes principales [8]:

1. Selección de parámetros, se selecciona un $m \in \mathbb{Z}$ y un polinomio f tal que $f(m) \equiv 0 \, mod \, n$

2. Definición de 3 bases de factores, se seleccionan las bases racional, algebraica y cuadrática.

3. Criba, se realiza una criba para generar las relaciones que construirán la matriz.

4. Procesamiento de relaciones: Se filtran las relaciones encontradas en el paso anterior, para encontrar las útiles y de no ser suficientes se debe realizar el paso 3 hasta que el número sea el requerido.

5. Construcción y solución del sistema lineal, con las relaciones encontradas se construye un sistema lineal que luego es resuelto.

6. Raíz cuadrada, con el resultado del punto 5 se generan dos cuadrados perfectos que se usan para factorizar n

A continuación explicaremos las secciones de una manera mas detallada

Selección de parámetros

En esta sección se debe elegir un polinomio f irreducible, con grado de acuerdo a esta tabla sugerida en [9] igual que el ejemplo a continuación

Longitud del número (dígitos en decimal)	Grado del polinomio (d)
50-80	3
80-110	4
110 en adelante	5

A continuación se debe elegir un número

$$m \simeq n^{\frac{1}{d}}$$

el polinomio a elegir será la expansión en base m de este n Ej.

Si $n = 45113$ entonces elegimos $d = 3$, según esto m podría ser un número cercano a 35, para los efectos del ejemplo elegimos 31, lo cual genera el polinomio

$$45113 = 31^3 + 15 \cdot 31^2 + 29 \cdot 31 + 8$$
$$f(31) = 45113$$
$$f(x) = x^3 + 15x^2 + 29x + 8$$

Definición de tres bases de factores[1]

Base racional

En esta base se elige un rango $[x, y]$,y se crea un conjunto de relaciones de la forma $(m(mod\ p), p)$ donde p es primo y $p \in [x, y]$

1 Adaptado de [8] y [9]

Base algebraica

En esta base se buscan parejas de la forma (r, p) donde p es primo y r es raíz del polinomio f (ver sección de selección de parámetros) modulo p , es decir:

$$\{(r, p): N \times N \mid primo(p) \wedge f(r) \equiv 0 \bmod p\}$$

Base cuadrática

En esta base se buscan parejas de la forma (r, q) donde q es primo y r es raíz del polinomio f modulo q, adicionalmente q es mayor que el mayor p de las parejas encontradas en la base racional, es decir que esta base encuentra las parejas de la forma

$$\{(r, p): N \times N \mid primo(p) \wedge f(r) \equiv 0 \bmod p \wedge q > max[p \in BR]\}$$

$BR: Base\ relacional$

Criba[2]

En esta sección se buscan generar las relaciones de la matriz, las relaciones son parejas de la forma (a, b) que cumplen las dos siguientes características

- $a + bm$ es suave con respecto a la criba racional ($R - Suave$)
- $a + b\theta$ es suave con respecto a la criba algebraica ($A - Suave$) y θ es raíz de $f(m)$.

Esta parte es de vital importancia para la paralelización del algoritmo porque se puede realizar de manera distribuida y además es la parte que requiere más tiempo.

Procesamiento de relaciones

En esta etapa se remueven de la lista de relaciones generadas en la etapa anterior, las parejas repetidas. Si las relaciones encontradas no son suficientes se debe realizar más criba.

Construcción y solución del sistema lineal[3]

En este paso se crea la matriz a partir de las relaciones filtradas del paso anterior. La matriz resultante posee las siguientes dimensiones

- Columnas: $2 + r + a + q$
- Filas: $2a + r + q$

r: Cardinalidad del conjunto de relaciones de la criba racional.

a: Cardinalidad del conjunto de relaciones de la criba algebraica.

q: Cardinalidad del conjunto de relaciones de la criba cuadrática.

La matriz es resuelta mediante el algoritmo de bloques Lanczos [9]

Raíz cuadrada[4]

La solución del paso anterior nos deja como resultado números de la forma

2 Adaptado de [8]
3 Adaptado de [8]
4 Adaptado de [9]

$x^2 = (a + bm)$

$y^2 = (a + b\theta)$

Teniendo estas respuestas se deben buscar las parejas que satisfagan

$x^2 \equiv y^2 \bmod n$

podemos encontrar los factores de n realizando

$f_1 = gcd(n, x + y)$
$f_2 = gcd(n, x - y)$

Con esto ya se tiene que

$n = f_1 \cdot f_2$

GGNFS una implementan de General Number Field Sieve

GGNFS es una implementan con licencia GPL del algoritmo General Number Field Sieve[17], realizada por Chris Monico profesor asistente del Departamento de Matemáticas y Estadística de la Universidad Texas Tech.

Mas que una caja negra para factorizar números grandes GGNFS consta de un grupo de aplicaciones, que se comunican a través de archivos y que son coordinadas mediante un script en Perl de nombre factlat.pl, el cual realiza las operaciones necesarias para llevar a cabo una factorización completa, sin embargo toda la puesta a punto del programa para factorizar un numero grande debe ser llevada a cabo por el usuario, por lo que para usar el programa de manera adecuada es necesario conocimiento acerca del algoritmo que este usa. A continuación presentaremos las etapas del GNFS, junto a su respectivo programa en GGNFS.

Etapa del algoritmo	***Programa de GGNFS***
Selección de parámetros	polyselect
Definición de bases de factores	makefb
Criba	gnfs-lasieveI13e
Procesamiento de relaciones	procrels
Construcción y solución del sistema lineal	matbuild y matsolve
Raíz cuadrada	sqrt

Berkeley Open Infrastructure for Network Computing (BOINC)

Historia

Las raíces de BOINC nacen del proyecto SETI@Home cuyo propósito es la "Búsqueda de Inteligencia Extraterrestre (siglas en inglés de SETI)" [10]. La idea principal consistía en aprovechar los recursos de computadores personales a través de Internet para procesar señales de

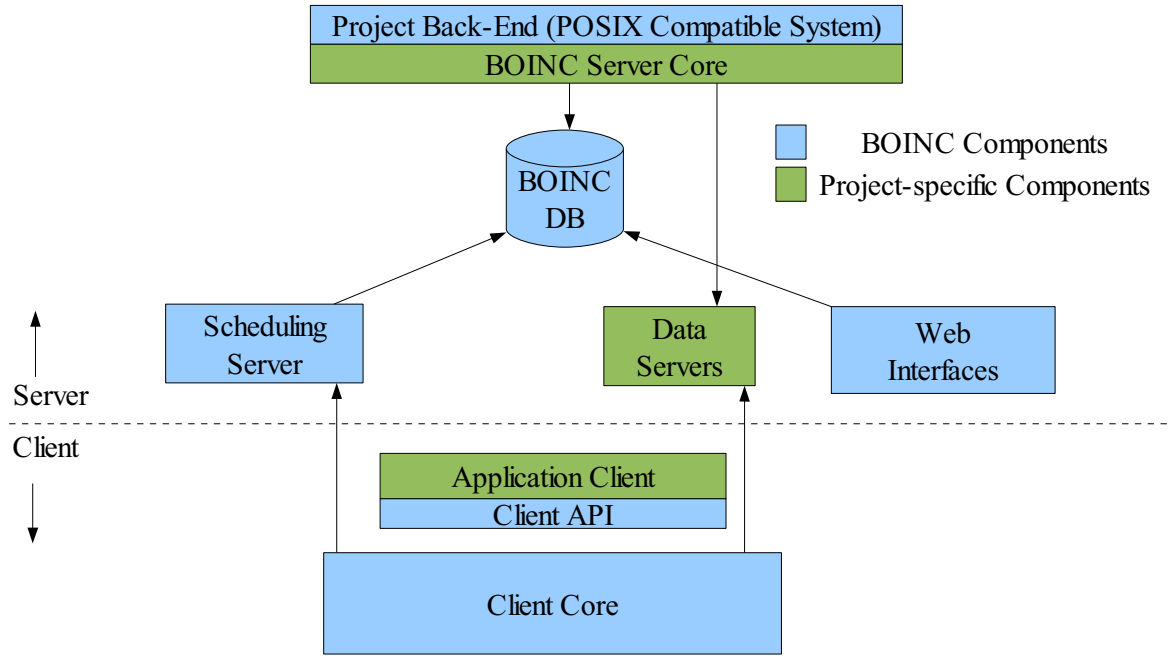
los grandes telescopios que detectan ondas de radio provenientes del espacio y buscan patrones para tratar de encontrar signos de vida inteligente fuera del planeta. Gracias a la gran acogida del proyecto se fundó el proyecto BOINC cuyo objetivo es "Avanzar el paradigma de computación de recursos públicos" [11]

Actualmente BOINC sirve de infraestructura de varios proyectos grandes a nivel internacional con varios cientos de miles de millones de usuarios. Su principal ejemplo SETI@Home cuenta actualmente con 560.000[5] usuarios aproximadamente. Otros ejemplos importantes son Einstein@Home, Climate Predicition y Roseta@Home

La capacidad computacional real de SETI@Home al día de hoy es de 272,219[6] TeraFLOPS. La capacidad potencial es varios ordenes de magnitud mayor, pero debido al paradigma voluntario solo una fracción de los ciclos potenciales se pueden utilizar en los proyectos. Aun así es uno de los computadores más potentes del mundo, y se podría decir que uno de los más económicos.

Descripción General

La arquitectura de BOINC es tipo Cliente-Servidor. Un esquema gráfico de cómo son los diferentes componentes del sistema se pueden ver a continuación en la ilustración Infraestructura de BOINC [12].



Drawing 1: Arquitectura de BOINC. Adaptado de [12]

El núcleo del servidor ejecuta bajo sistemas tipo UNIX. En particular está probado y listo para funcionar en sistemas con kernel Linux. Para manejar la persistencia de los trabajos y el estado del sistema BOINC utiliza una base de datos MySQL. La comunicación con los clientes es totalmente vía HTTP por el puerto 80 intermediado por un servidor HTTP Apache con Server Side Scripting.

El sistema se ejecuta por medio de varios programas independientes que interactúan entre sí por medio de los mecanismos de memora compartida de UNIX, la base de datos, y archivos en la estructura de carpetas del proyecto.

5 http://www.boincstats.com/index.php?pr=bo
6 http://www.boincstats.com/stats/project_graph.php?pr=sah

Funcionamiento

La infraestructura de BOINC se encapsula en *proyectos*. Dentro de las definiciones generales se asume que un servidor es equivalente a un proyecto, aunque en la realidad se sabe que el servidor es un conjunto de aplicaciones que corren en paralelo y se comunican entre ellas.

Un mismo proyecto puede ser escalado a cuantos servidores se desee. La forma para lograr el escalamiento es asignando los diferentes servicios en máquinas diferentes. Para escalar proyectos a más máquinas que el número de servicios es necesario replicarlos a través de varias máquinas. BOINC cuenta con un sistema de replicación en el cual a cada máquina se le asignan trabajos x tal que $job_{id} \bmod N = x$, siendo N el número total de máquinas que replican el servicio. De esta forma todas las máquinas se subdividen el trabajo sin hacerlo redundante. Para el resto del funcionamiento del sistema se asumirá que todo el proyecto se ejecuta en una sola máquina.

En cada proyecto de BOINC está auto contenido el núcleo del servidor, esto logra garantizar la independencia entre varios posibles proyectos de una única máquina. Un proyecto está identificado por una única URL, llamada *master URL*. Esta sirve como punto de referencia para nuevos clientes que deseen donar sus máquinas, y en la mayoría de casos también es la puerta de entrada de las transacciones con los clientes existentes.

Cada proyecto tiene un propósito, pero para lograrlo se debe subdividir en varias *aplicaciones*. La forma más fácil de entenderlas es que cada aplicación es un programa específico, cuya única comunicación con otros programas es la entrada y salida de archivos.[7] Cada aplicación es descargada automáticamente por los clientes de BOINC y se mantiene actualizada para cada trabajo, de forma que siempre se trabaje con la última versión cada vez que se empieza un nuevo trabajo.

BOINC maneja un sistema de firmas digitales para garantizar en todo momento la integridad y la autenticidad de las aplicaciones. De esta forma se vuelve imposible para un atacante inyectar código a los clientes para sus propios propósitos. Las firmas pueden ser generadas por el servidor o preferiblemente generadas en una máquina no conectada a la red. Cada cliente posee el certificado de la autoridad certificadora que autentica las firmas digitales de los proyectos.

Los trabajos se dividen en unidades de trabajo llamadas *workunits*. Una unidad de trabajo corresponde a una ejecución de una aplicación con un conjunto de parámetros completamente definidos. Los parámetros de cada trabajo pueden llegarle al cliente dentro de la especificación del trabajo o como archivos adjuntos. Una tarea grande se divide en varios subproblemas o *workunits,* los cuales se procesan en los clientes y devuelven sus resultados para luego ser reunidos y asimilados posteriormente.

Debido al paradigma de computación voluntaria (public resource computing) en el cual se basa BOINC, no es posible confiar totalmente en los clientes. Para evitar estos inconvenientes, la infraestructura cuenta con un sistema de réplica de trabajos (redundant computing); en el cual a cada *workunit* se le adjudican varios resultados redundantes, y el servidor se encarga de encontrar un resultado común o *canonical result*. Este resultado es el que luego se asimila como resultado aceptado y confiable. Cada unidad de trabajo se define con un número de resultados esperados y un mínimo número de resultados para el *canonical result*. De esta forma es perfectamente posible definir *workunits* sin redundancia en caso que se cuenten con clientes confiables.

Cuando un cliente realiza una petición de trabajo este incluye la carga que desea tener, así como la cantidad de recursos de memoria y disco disponibles. Cada cliente BOINC mantiene un punto de referencia de la capacidad computacional (operaciones de entero y de punto flotante por segundo) el cual se usa para comparar con la cantidad de trabajo enviada. A cada *workunit* se le

7 Esta restricción puede ser violada utilizando el mecanismo de trickle messages de BOINC

definen el estimado y el máximo de operaciones de punto flotante. El estimado se usa para distribuir los trabajos a los clientes que tengan capacidad de realizarlo (según las peticiones que hagan) y el máximo es usado por los clientes para abortar los trabajos que puedan quedarse ejecutando eternamente.

Arquitectura del servidor

El servidor de BOINC se subdivide en cinco servicios principales, algunos independientes del proyecto y otros específicos a cada una de las aplicaciones.

Servicios Independientes de las Aplicaciones

Servidor de carga y descarga

Este servicio es el encargado de administrar las descargas de las aplicaciones y archivos adjuntos necesarios de los *workunits,* y de manejar las subidas de archivos resultados al servidor para luego ser validados y verificados.

La primera tarea la cumple principalmente el servidor HTTP Apache redireccionado por medio de *Aliases*. La segunda se ejecuta por medio de un Server Side Script (file upload handler) con la ayuda del servidor HTTP Apache.

Programador de trabajos

La programación de trabajos es una de las principales tareas de BOINC, es lo que lo diferencia de un simple sistema de información. Este servicio lo ejecuta otro Server Side Script con la ayuda del servidor HTTP Apache. El programador de trabajos también es el encargado de manejar los *RPC's* entre los clientes y el servidor.

Alimentador de Trabajos

El propósito principal del alimentador de trabajos es pasar la información de la base de datos al programador de trabajos a través de un segmento compartido de memoria. Este también se encarga de manejar las prioridades de los trabajos y las aplicaciones.

Administrador de Transacciones

Este servicio se encarga de administrar los estados de los *workunits*. Esto incluye enlistar nuevos resultados cuando hayan ocurrido errores.

Borrador de Archivos

Con este servicio se busca simplemente liberar el espacio en el servidor ocupado por los *workunits* y *results* ya concluidos.

Servicios específicos para cada Aplicación

Estos servicios deben ser programados para cada aplicación. BOINC provee un conjunto de interfaces de ejemplo para facilitar la labor de los desarrolladores.

Validador

El validador se encarga de verificar que haya quorum en los resultados y de asignarles un

crédito a los mismos. Esto quiere decir encontrar un consolidado de todos los resultados obtenidos para un *workunit*. Varia según el tipo de aplicación. Algunas opciones son que los resultados del quorum sean iguales bit a bit, o el caso en el cual no es necesario o posible hacer una validación (por ejemplo si es información basada en alguna semilla aleatoria). Estas dos alternativas se pueden utilizar sin necesidad de programar gracias a dos validadores que vienen con la distribución estándar de BOINC.

Asimilador

El asimilador es uno de los servicios más importantes del BOINC. Con él se pueden tomar las decisiones pertinentes una vez terminado un *workunit* o un conjunto de estos. Cualquier acción es posible ya que se debe implementar personalizado para cada aplicación. Entre las posibles opciones del Asimilador están generar reportes o emails de terminación, generar más trabajo, o agrupar varios *workunits* y enviarlos para postproceso en algún servidor.

Arquitectura del Cliente

El cliente BOINC aparenta y actúa como un solo programa, igual que el servidor, pero en realidad es un conjunto de programas que se comunican entre ellos. El siguiente diagrama[8] ayuda a entender mejor los diferentes componentes del cliente.

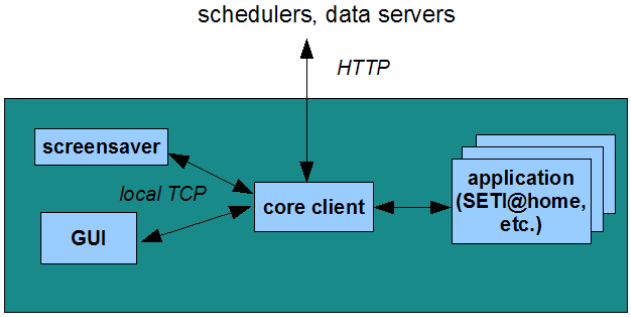


Ilustración 3: Arquitectura del cliente BOINC

El núcleo del cliente se ejecuta en un proceso independiente. En máquinas Windows este proceso puede ejecutarse como servicio o a nivel de usuario según el tipo de instalación y las necesidades. Se encarga de todas las funciones de BOINC, administrar las tareas, descargas, estadísticas, errores, seguridad, comunicación con los servidores, etc. Bajo su administración se ejecutan cada una de las aplicaciones según los proyectos a los que se haya añadido el cliente.

Para su administración existen dos interfaces gráficas, una avanzada y otra simple. Ambas son el mismo programa corriendo bajo diferentes opciones. Por medio de ellas se pueden controlar las funciones del cliente como por ejemplo no pedir más trabajo, añadirse a nuevos proyectos y revisar el estado del sistema: descargas, trabajos, crédito, espacios, mensajes, etc. Se comunica con el núcleo por medio de sockets TCP locales.

El protector de pantalla es una de las principales funciones de BOINC que lo diferencian de otras infraestructuras para computación distribuida. Es de especial interés debido al factor publicitario que este puede generar en los clientes. Además, sirve para informar al usuario que se están ejecutando labores de los proyectos a los que se ha añadido. Igual que la interfaz gráfica se comunica con el núcleo utilizando sockets TCP locales. Este puede ser personalizado para cada

8 Tomado de http://boinc.berkeley.edu/client.png

aplicación por medio del API gráfico de BOINC y OpenGL.

Interacción entre el cliente y el Servidor

A las comunicaciones entre los clientes y el servidor se les llaman *Remote Procedure Calls (RPC)* que en práctica son conexiones TCP por el puerto 80 entre el cliente y el servidor. Los mensajes se envían en formato XML con protocolos específicos para BOINC (No es SOAP). El siguiente diagrama[9] muestra de manera gráfica la secuencia de pasos en que interactúan el cliente y el servidor.

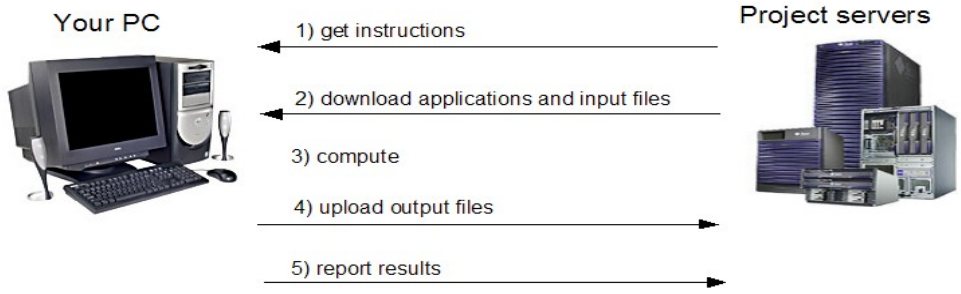


Illustration 1: Interacción entre el cliente y el servidor BOINC .

1. El cliente recibe la definición del *workunit* que debe realizar
2. Se descargan los archivos ejecutables (en caso que haya una nueva versión) y los archivos adjuntos a la tarea.
3. Se ejecutan las aplicaciones en los clientes.

   Nota: Debido una vez más a la naturaleza voluntaria del BOINC no es posible predecir exactamente cuando los clientes pedirán trabajos, ni cuando se terminarán. En un caso particular el paso tres *calcular* podría tomar más tiempo del normal debido al trabajo realizado por los usuarios que utilizan la máquina para sí mismos.
4. Al terminar el cliente sube los archivos al servidor

   Nota: Para evitar ataques al servidor cada resultado viene firmado con una llave pública especial diferente a la de las aplicaciones, de esta forma solo los resultados con firma válida pueden subir archivos, evitando subidas masivas de archivos al servidor.
5. Si finaliza correctamente la carga de archivos resultado al servidor se genera un nuevo *RPC* reportando lo resultados, reclamando crédito y pidiendo nuevos trabajos.

9 Tomado de http://boinc.berkeley.edu/comm_simple.png

Desarrollo

A continuación explicaremos el desarrollo del proyecto, empezando por las diferentes posibilidades estudiadas para escoger el algoritmo; luego trataremos el estudio de las implementaciones disponibles y cómo paralelizarlo. Una vez hayamos mostrado el análisis de los algoritmos de factorización entraremos en el tema de las posibles infraestructuras de computación paralela y de cómo implementar la paralelización del algoritmo en la infraestructura escogida. Finalmente nombraremos los detalles de la instalación masiva y las pruebas realizadas para evaluar la eficiencia del algoritmo en paralelo.

¿Que algoritmo usar?

Al inicio del proyecto no teníamos claro que algoritmo debíamos usar. Según [13] existen varios algoritmos modernos para la factorización de números grandes, algunos mucho mejores que otros. Entre ellos podemos nombrar el $Pollard(\rho-1), CFRAC, MPQS, ECM, NFS$ y $GNFS$. El último es el actual campeón para números de más de 100 cifras decimales y por lo tanto fue el que elegimos para trabajar. Al inicio del proyecto realizamos una implementación del algoritmo de Pollard con el cual iniciamos nuestro camino hacia el GNFS.

Implementaciones

Para la implementación evaluamos la posibilidad de realizarla nosotros mismos, sin embargo encontramos que esto se salía de los alcances del proyecto, debido a que requiere demasiado tiempo. Encontramos cuatro implementaciones disponibles públicamente en Internet.

- GGNFS: Realizada por Chirs Monico, Profesor Asistente Texas Tech University.
- Factor-by-GNFS: Realizada por Chris Card.
- pGNFS: Realizada por Per Leslie Jensen, Tesis de maestría en Ciencias de la Computación, Universidad de Copenhague.
- Msieve: Realizada por Jason Papadopolus.

Cada una de las implementaciones tiene sus fortalezas y debilidades. Empezando de abajo hacia arriba, la implementación de Msieve tiene a favor que es totalmente auto contenida, por lo cual no necesita ninguna librería adicional para ser compilada y ejecutada. Es multiplataforma y ha sido probada en sistemas tipo Windows, Linux, Mac, y AIX. Su principal debilidad es el rendimiento que es bastante inferior a las demás implementaciones. De hecho el mismo autor limita su uso a 120 cifras decimales y reconoce su inferioridad frente a otras implementaciones[14].

La implementación de Per Leslie Jensen pGNFS es bastante joven. Hasta el momento no se ha probado con números superiores a 60 dígitos [15], por lo cual no cumple con los requisitos mínimos deseables. Otra debilidad que presenta esta implementación son las librerías de números grandes que utiliza NTL y GiNaC, en particular la segunda no está disponible para sistemas Windows [16].

Factor-by-GNFS es una implementación poco probada hasta el momento. Sin embargo las referencias de otras personas indican que es una buena opción. También es auto contenido pero no es claro si tiene soporte para sistemas UNIX. Cuenta con muy poca documentación por lo que no es muy popular.

GGNFS [17] es la mejor opción para nuestros objetivos. Cuenta con versiones para Windows

y Linux. Tiene una comunidad de personas dispuestas a ayudar, junto con su creador. Es la distribución libre más probada y está pensada para paralelizarse, por lo cual todos los pasos se hacen en programas diferentes que se comunican únicamente por archivos y parámetros de ejecución.

Paralelización

La mayor parte del algoritmo GNFS se dedica al cribado. Este paso consume aproximadamente el 70 u 80 por ciento del tiempo total de ejecución del algoritmo [8]. Consideramos que éste era el punto crítico para paralelizar. La resolución de la matriz también es un paso que consume bastante tiempo, sin embargo los algoritmos para paralelizarlo tienen un radio de input/output muy grande [18] que dificultan la tarea en la infraestructura que teníamos disponible.

En particular para la implementación de GGNFS la criba se realiza mediante el programa *ggnfs-lasieve4I13*. Este recibe los parámetros mediante un archivo de texto, tradicionalmente con extensión *job.* A continuación se muestra un ejemplo de un *job* para el RSA-110 [17].

```
n:35794234179725868774991807832568455403003778024228226193532908190484670252364677411513516111204504060317568667
skew: 18132.78
# norm 2.01e+15
c5: 77700
c4: 4651351957
c3: -85642705009916
c2: -240017091501865032
c1: 12706215374564843828616
c0: -18727334707397357259852480
# alpha -6.18
Y1: 17432622119
Y0: -856403247374965385213
# Murphy_E 1.01e-09
# M 126346411885860100672422577375964831034983458720765380 0
3789135795080537828937172615657373569268546264716586221
type: gnfs
rlim: 3200000
alim: 1599999
lpbr: 27
lpba: 27
mfbr: 50
mfba: 50
rlambda: 2.6
alambda: 2.6
q0: 1675601
qintsize: 5399
```

Ilustración 1: Archivo de trabajo para ggnfs-lasieve para RSA-110

Los parámetros importantes son *q0* que indica el punto inicial de la criba y *qintsize* que indica cuanto se debe cribar. Lo único que se debe modificar es el *q0*, que debe variar para cada trabajo. El *qintsize* es posible variarlo también, pero resulta más práctico dejarlo igual y así se logra mantener las unidades de trabajo de complejidad constante.

El resultado de la criba lo deja en un archivo de texto generalmente llamado *siever.out* que puede llegar a varios Megabytes fácilmente. Para integrar varias respuestas hay dos maneras: la primera es reunir todas las respuestas en un solo archivo para luego ser procesado por el procesador de relaciones, y la segunda es adicionarlas con el procesador de relaciones que las convierte en binario para construir la matriz. Lo ideal es llegar a un punto medio, debido a que juntar todas las respuestas para luego procesar las relaciones con archivos muy grandes, puede llegar a ser muy costoso en memoria y en tiempo, pero hacer muchas veces el procesado de relaciones puede desperdiciar tiempo debido a que se deben cargar las relaciones anteriores.

El paso de procesar las relaciones para pasarlas de texto a binario se puede ir realizando

paralelamente a la criba, esto ayuda a maximizar el tiempo porque la mayor parte de las relaciones estarían procesadas antes de terminar la criba.

¿Que tipo de infraestructura usar?

El proceso de selección de la tecnología a emplear, se realizó basado en los siguientes requerimientos:

- Constituir un Grid completamente funcional con un solo computador de escritorio.
- Permitir escalabilidad.
- Aprovechar el tiempo libre de los computadores durante las horas de servicio de las salas.
- Vincular como trabajadores del Grid computadores que se encuentren fuera de las salas (profesores, área administrativa).
- Soportar dinámicamente la adición y el retiro de trabajadores al Grid.
- Soportar múltiples plataformas de software, debido al cambio de sistema operativo en las noches propuesto por el Grid-Uniandes.
- Soportar el manejo de errores y recuperación de los trabajadores.
- Permitir la administración remota tanto de los trabajadores como de la infraestructura administrativa del Grid
- Permitir un alto nivel de configuración para no acaparar los recursos de computo de la universidad de manera desmesurada.
- Permitir la recaudación de estadísticas de los trabajos realizados.
- No comprometer la seguridad de los trabajadores del Grid
- Brindar una interfaz gráfica en los trabajadores para enterar a la comunidad uniandina del proyecto.

Infraestructuras disponibles

Al iniciar nuestro proyecto en la universidad de los Andes, se esta experimentando con Sun Grid Engine y gLite para soportar la infraestructura del Grid-Uniandes, sin embargo estas tecnologías tenían ciertos inconvenientes para nuestro proyecto como:

- De trabajar sobre estas infraestructuras seriamos usuarios y no administradores del Grid, por lo cual estaríamos sujetos a las políticas de los administradores, lo cual no nos permitiría el alto nivel de configuración que deseábamos a nivel de servidor y trabajadores.
- Según lo que investigamos sobre ambas tecnologías, los frameworks de programación que proveían no eran las mas adecuadas para nuestro proyecto ya que se necesitaba realizar manualmente el manejo de errores y la recuperación ante fallas, lo cual hubiese incrementado el trabajo a realizar dramáticamente
- El requisito de poner a funcionar los trabajadores durante las horas de servicio de las salas no es posible debido al esquema de asignación de tareas empleado por estas tecnologías; además en gLite no existe un cliente para plataforma Windows.
- El requisito de la interfaz gráfica en modo de salva pantallas no era cubierto por

ninguna de las dos tecnologías

- Sun Grid Engine es un producto propietario, lo cual no es deseable en los ambientes académicos

Estos inconvenientes, nos llevaron a investigar alternativas entre las cuales BOINC fue la mejor solución por las siguientes razones:

- El servidor puede funcionar en un solo computador de bajo desempeño, empleando herramientas libres. Además se pueden distribuir los servicios para permitir escalabilidad.
- Por su filosofía de computación distribuida voluntaria, BOINC solo realiza trabajo cuando el computador no se esta usando, lo cual es óptimo para instalar en las salas de estudiantes.
- El nivel de configuración de los clientes es el necesario para nuestro proyecto ya que permite configurar: el número de procesadores a usar por trabajador, tiempo de espera antes de iniciar trabajo y horas del día en las que el trabajador va a trabajar, entre otros.
- El cliente es multiplataforma y realiza de manera transparente para los trabajadores la adquisición de los programas, las tareas a ejecutar y el envió de resultados. Además la construcción de interfaces para el cliente es llamativa y logra cumplir el requerimiento de llamar la atención de la comunidad uniandina.
- Los clientes pueden añadirse o removerse del proyecto fácilmente
- El manejo de errores y la recuperación ante fallas es automática
- El sitio de estadísticas permite tener información detallada del proyecto en todo momento de forma automática
- La autenticación del servidor con sus clientes es fuerte para evitar problemas de seguridad.

Implementación de la Criba de GGNFS en BOINC

Para implementar la criba paralela en BOINC se utilizó la idea de tener varios archivos con diferentes puntos de inicio. Cada trabajo requería un archivo *job* diferente por lo cual se debería expresar en BOINC como un *workunit* diferente.

Primero fue necesario adaptar el programa *gnfs-lasieve* para comunicarse con BOINC. Habían tres puntos específicos en los cuales era necesario comunicar *gnfs-lasieve* con el cliente de BOINC: El primero era la inicialización: todas las aplicaciones de BOINC deben iniciarlizar el sistema mediante un llamado a la función *boinc_init* que hace parte del API estándar.

El Segundo era la lectura del archivo *job*. Este archivo llegaba adjunto a cada *workunit* por lo cual su nombre físico no sería idéntico pero su nombre lógico dentro de la aplicación si debería serlo. Para este caso BOINC cuenta con una función que permite resolver los nombres físicos a nombres lógicos: *boinc_resolve_filename*. Cambiamos los llamados a funciones para manejo de archivos por las de BOINC.

El tercer paso era similar al segundo pero para la salida. El nombre del archivo lógico era el mismo para todas las aplicaciones pero debería cambiar físicamente para que no hubieran conflictos en el servidor. Para solucionar esto BOINC cuenta con el result_template.xml el cual hace un mapeo entre los archivos producidos por la aplicación y los resultados que se desean subir al servidor. BOINC se encarga automáticamente de nombrar estos archivos de salida de tal forma que sean consistentes con la identificación del *workunit* y del número del intento que se esté realizando (en caso de errores o redundancia).

Un paso adicional no obligatorio que hicimos fue el reporte del porcentaje completado. Este número se presenta a los usuarios en el protector de pantalla y por lo tanto consideramos que era bastante importante para nuestro objetivo de enterar a la comunidad uniandina del Grid. De otra forma siempre se vería que el trabajo está en 0 por ciento y esto puede llegar a confundir a las personas.

Una vez adaptada y compilada la aplicación para la infraestructura procedimos a crear los *jobs* para el caso del RSA-110. Escogimos este número debido a que su tiempo de factorización es mediano con respecto a nuestro tiempo de pruebas. Así podíamos correrlo varias veces en caso de errores y podíamos comparar con una ejecución no paralelizada del mismo.

La creación de los *jobs* y los *workunits* para el servidor se realizó mediante scripts escritos en Perl, genéricos para cualquier *job* y *workunit.* El proceso de factorización no es automatizado. Existen varios scripts, algunos de BOINC otros de nosotros, que ayudan al proceso; pero sigue siendo necesaria la acción de las personas involucradas en todas las etapas del algoritmo. Nos parece que no es muy buena idea ocupar tiempo en automatizar estos procesos debido a que sería necesario monitorearlos muy atentamente y aun así los números grandes que vale la pena factorizar requieren muy poco tiempo administrativo en relación al tiempo total del algoritmo.

Instalación masiva

El personal administrativo del MOX nos prestó una gran ayuda en la instalación masiva de los clientes. Para cumplir mejor nuestro objetivo publicitario decidimos crear un protector de pantalla personalizado que explicara en qué se estaban utilizando las máquinas y que concientizara a las personas de la existencia del Grid. La imagen mostrada en el protector de pantalla se muestra a continuación. Queremos dar un agradecimiento especial a Danilo Perez por su gran ayuda para realizar esta imagen

Ilustración 5: Protector de Pantalla de los clientes del GRID-MOX-UNIANDES

Ademas, en la búsqueda del objetivo anterior decidimos personalizar totalmente los clientes, de tal forma que no se encontrara la palabra BOINC. Esta se reemplazó por la palabra GRID-MOX-UNIANDES, que esperamos identifique este Grid dentro de la universidad. Así mismo se reemplazaron todos los iconos de BOINC por iconos de cabras que representan la mascota del GRID-MOX-UNIANDES y de la Universidad. Todas las imágenes que tomamos de Internet son de uso comercial libre y por lo tanto adecuadas para nuestro proyecto de interés académico.

Creamos un instalador personalizado para el Grid que permitiera la instalación del cliente minimizando la interacción con el usuario, por ejemplo instalando el programa como un servicio de Windows sin la necesidad de tener un usuario específico. Así facilitamos la instalación para los monitores de las salas.

Diseño de Experimentos

Para probar el algoritmo paralelo decidimos usar el número RSA-110. Este número tiene algunas propiedades que lo hacen una buena escogencia en este tipo de pruebas como:

- Se conoce un buen polinomio para factorizarlo
- Hay pruebas de muchas personas con diferentes tiempos
- Es relativamente lento en una sola máquina, aproximadamente 24 horas.
- Permite realizar varias pruebas durante los días disponibles
- La resolución de la matriz es relativamente corta, aproximadamente 2 horas.

En el momento de realizar la prueba contábamos con 151 máquinas disponibles para el Grid, entonces decidimos hacer este mismo número de trabajos, de tal forma que cada máquina hiciera un pedazo y así minimizar la pérdida de tiempo debido a factores de comunicación y descarga de archivos.

Ejecutamos la factorización completa en Guaica, uno de los servidores más potentes de la universidad[10], para luego compararlo con los tiempos de factorización en paralelo. Decidimos procesar las relaciones y resolver la matriz en Guaica debido a que el administrador del Grid Cronos no cuenta con la memoria RAM necesaria para resolver este número, además así podríamos tener números comparables para la resolución de las matrices con criba paralela y secuencial.

Planeamos la criba para que se realizara en las horas de la noche así contábamos con la totalidad de la capacidad del Grid y minimizábamos las demoras producidas por el uso regular de las máquinas por parte de los estudiantes. Los trabajos (*workunits*) se diseñaron de forma que no presentaran computación redundante, debido a que todos nuestros clientes son confiables.

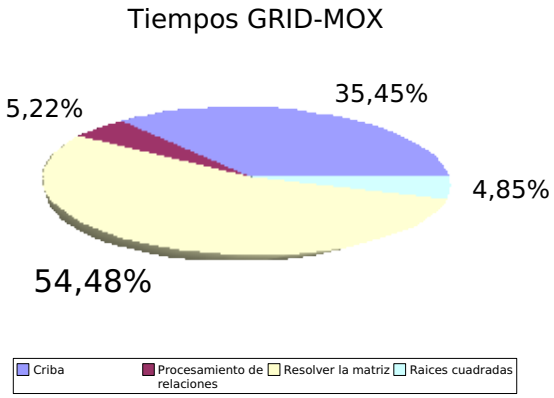
Resultados

A continuación mostramos los tiempos de cada una de las etapas del algoritmo en cada uno de los experimentos

	Guaica (horas)	GRID-MOX (horas)
Criba	21,83	0,95
Procesamiento de relaciones	0,16	0,14
Resolver la matriz	1,24	1,46
Raices cuadradas	0,13	0,13
Total	23,36	2,68

A continuación se encuentran los porcentajes de tiempo de cada etapa en cada uno de los experimentos.

10 Guaica cuenta con 8 Gigabytes de memoria RAM, 4 procesadores AMD Opteron 850 de 2400 Mhz y 380 Gigabytes de disco duro SCSI.



Gráfica 1: Tiempos de ejecución GRID-MOX en Paralelo

Gráfica 2: Tiempos de Ejecución Guacia No Paralelo

Durante la ejecución distribuida de la criba participaron 83 computadores.

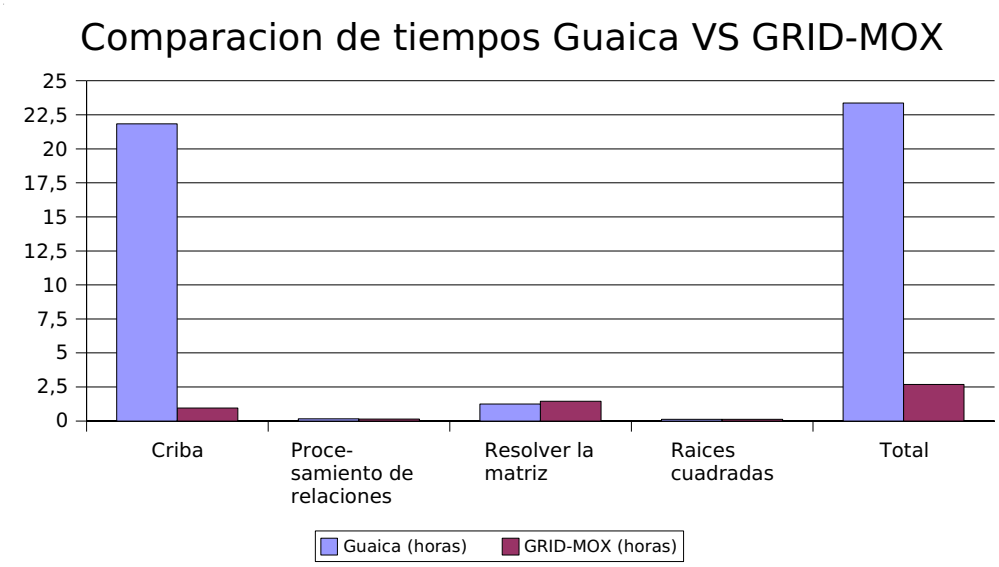
Las mediciones de los tiempos se realizaron así:

- Para las etapas que se ejecutan en un solo computador se tomaron los tiempos registrados por el GGNFS.
- Para la etapa de criba distribuida la medida del tiempo de ejecución se calculo con esta formula

$$T = T_s - T_r$$
$$T_s : El\ tiempo\ en\ que\ se\ envio\ la\ primera\ unidad\ de\ trabajo$$
$$T_r : El\ tiempo\ en\ que\ se\ recibio\ la\ ultima\ respuesta$$

Nota: La razón por la que se calcula el tiempo de esta manera, es que no se puede obligar a los clientes de Boinc que pidan trabajo, por esta razón tomamos como punto de partida el primer momento en que uno de los nodos pidió trabajo.

Los datos de tiempo



Gráfica 3: Comparación de tiempos Guacia Vs GRID-MOX

Como se puede ver en la gráfica el tiempo total de ejecución en el Grid es mucho menor que en Guaica, debido a que se paralelizó la parte que mas toma tiempo del algoritmo, de hecho la criba se ejecuto 22.98 veces mas rápido de la manera distribuida.

Aunque estos resultados son importantes y muestran el gran poder computacional disponible en las salas, la naturaleza voluntaria de BOINC obliga a esperar que los clientes pidan tareas. Esto, sumado al menor poder de computo de los computadores de las salas y a la diferencia de arquitecturas de los procesadores, son razones por las cuales los 83 computadores no lograron realizar el trabajo de criba en una relación mas cercana al optimo de 1/83 del tiempo con respecto a Guaica. Esto se puede apreciar en que la suma de los tiempos de ejecución de los procesos en cada uno de los clientes fue de 35.89 horas, lo cual es bastante superior a las 21.83 horas de Guaica ante una carga de trabajo similar.

Conclusiones y Sugerencias

Logramos escoger la mejor alternativa para la infraestructura de computación distribuida que requería el problema de factorización de números grandes. BOINC cumple con todas las características deseables para muchos problemas, es escalable, liviano, multiplataforma y no requiere clientes dedicados. Sin embargo no es ideal para trabajos relativamente cortos debido a que se debe esperar a que los voluntarios pidan tareas, lo que puede tomar mucho más tiempo que en sistemas tipo push. Tampoco es ideal para problemas que requieran un gran número de aplicaciones personalizadas debido a que se debe ajustar el programa para funcionar en BOINC. Cabe anotar que existe la forma de ejecutar aplicaciones legado que no poseen la capacidad de comunicarse con BOINC, pero de todas formas se debe realizar un proceso de automatización e integración de *workunits* para la generación y asimilación de trabajos para que este tipo de sistemas sean acoplados a BOINC.

Factorizar RSA-704 es posible con bastante dedicación pero con la implementación actual de GGNFS se corren riesgos debido a las limitaciones intrínsecas que posee el código. Sin embargo vale la pena hacer el intento tanto por la experiencia de administrar la computación distribuida como por el reto académico que representa el problema a nivel mundial.

Nos dimos cuenta que hay un gran desperdicio de capacidad computacional en las noches al apagar las salas. Sería muy bueno extender este tipo de proyectos a otros departamentos y que toda la comunidad se beneficie de sus propios recursos.

También gracias a las estadísticas generadas por los clientes de BOINC pudimos calcular la utilización instantánea promedio de cada computador de las salas. Cada equipo está siendo utilizado 87.91 por ciento del tiempo, una cifra bastante alta y que explica los problemas de algunos estudiantes para encontrar disponibilidad en los recursos.

Agradecimientos

Deseamos agradecer especialmente a Milton Quiroga por la ayuda y el apoyo incondicional brindados al proyecto, a Danilo Pérez por su incansable colaboración y consejos, al equipo administrativo del MOX por su gran ayuda en la instalación masiva de los clientes y a David Anderson creador y director del proyecto BOINC que nos ayudó con problemas técnicos y sugerencias para nuestro proyecto.

Referencias

1. RSASecurity, RSA-640 is factored!. [en linea] Disponible en: http://www.rsasecurity.com/rsalabs/node.asp?id=2964

2. Wikipedia contributors , RSA Factoring Challenge. [en linea] Disponible en: http://en.wikipedia.org/wiki/RSA_Factoring_Challenge

3. W. Stallings, Cryptography and Network Security: Principles and Practice (3rd Edition), Prentice Hall, 2002

4. Wikipedia contributors , RSA algorithm. [en linea] Disponible en: http://en.wikipedia.org/wiki/RSA_algorithm

5. Wikipedia contributors , Integer factorization. [en linea] Disponible en: http://en.wikipedia.org/wiki/Integer_factorization

6. M. O. Saarinen, Lecture 6: Cryptanalysis of public-key algorithm Helsinki University of technology[en linea] Disponible en: http://www.tcs.hut.fi/Studies/T-79.159/2004/slides/L6.pdf

7. P. Leslie, RSA encryption using other groups than Z/Zn?, [en linea] Disponible en: http://www.pleslie.dk/DOC/RSA-groups.pdf

8. L. Xu et al. Factoring large integers using parallel General Number Field Sieve, http://scholar.google.com.co/scholar?hl=es&lr=&safe=off&q=cache:1Y6oVHPHpBYJ:cs.stfx.ca/~lxu/download/PDPTA-05.pdf+GNFS+factoring

9. M. E. Briggs, An Introduction to the General Number Field Sieve, [en linea] Disponible en: scholar.lib.vt.edu/theses/available/etd-32298-93111/unrestricted/etd.pdf

10. Science Space Lab, Seti@home Universidad de Berkeley, [en linea] Disponible en: http://setiathome.berkeley.edu/

11. D.P. Anderson, BOINC: A System for Public-Resource Computing and Storage. [en linea] Disponible en: http://boinc.berkeley.edu/grid_paper_04.pdf

12. J.A Lopez, BOINC Architecture and basic principles. [en linea] Dispponible en: https://uimon.cern.ch/twiki/pub/LHCAtHome/LinksAndDocs/boincciemat06.pdf

13. H. Riesel, Prime Numbers and Computer Methods for Factorization (2nd Edition), Birkhauser, 1994

14. J. Papadopolus, Msieve, [en linea] Disponible en: http://www.boo.net/~jasonp/qs.html

15. P. Leslie, pGNFS, [en linea] Disponible en: http://www.pgnfs.org/

16. Ginac authors, Ginac, [en linea] Disponible en: http://www.ginac.de/Download.html

17. C. Monico, GGNFS suite, [en linea] Disponible en: http://sourceforge.net/projects/ggnfs\

18. R. P. Brent, Recent progress and prospects for integer factorisation algorithms. [en linea] Disponible en: http://citeseer.ist.psu.edu/cache/papers/cs/16088/ftp:zSzzSzftp.comlab.ox.ac.ukzSzpubzSzDocumentszSztechreportszSzTR-3-00.pdf/recent-progress-and-prospects.pdf