

Universidad Nacional de Córdoba
Facultad de Matemática, Astronomía y Física



Especialidad en Sistemas y Servicios Distribuidos

Trabajo Final Integrador

Tema: Análisis y Mejoras de Sistemas de Cómputo Voluntario

Autores: Ing. Javier Jorge
Lic. Eduardo Sanchez

Director: Ing. Pablo Passera

Año 2011

Esta página ha sido dejada intencionadamente en blanco.

Agradecimientos

A la Universidad Nacional de Córdoba, al gobierno de la Provincia de Córdoba y a Intel por financiar y promocionar Especializaciones, permitiendo mejorar continuamente la formación de recursos humanos.

A todos nuestros compañeros de Cohorte; en especial a Agustín y Mónica que siempre nos instaron a seguir con la confección del trabajo integrador.

A nuestras familias.-

A todos los que nos quieren.-

A todos los que de alguna u otra manera...

Índice de contenido

Resumen.....	7
Abstract.....	8
Introducción	9
Cómputo Voluntario.....	9
Cómputo Voluntario Vs. Cómputo en red (Grid Computing).....	9
Historia.....	10
Tecnologías y Frameworks en Cómputo Voluntario.....	11
BOINC.....	13
Arquitectura.....	13
Lo que se ejecuta en el cliente.....	14
Lo que se ejecuta en el servidor.....	14
Funcionamiento de BOINC.....	15
Datos interesantes a tener en cuenta.....	15
Composición Interna del Servidor de Tareas.....	16
Ciclo de vida de un trabajo.....	17
XtremWeb.....	18
Arquitectura.....	18
XtremWeb-CH.....	19
Arquitectura.....	20
Componentes.....	20
El coordinador.....	20
El trabajador.....	20
El almacén.....	20
Xgrid.....	22
Componentes.....	22
Características sobresalientes.....	23
Sistema de archivos compartido.....	23
Grid MP.....	25
Seguridad.....	25
Arquitectura.....	26
Componentes de Grid MP.....	28
Program Loader.....	28
Servicio MGSI - RPC.....	28
Consola MP.....	28
Base de datos.....	28
Agente MP.....	28
Service Manager.....	28
Realm Service.....	29
Poll Service.....	29
Dispatch Service.....	29
Servicio de Archivos.....	29
SLINC.....	30
Arquitectura.....	30
XWHEP.....	32

Componentes.....	33
Servicio de Archivos y Seguridad.....	34
Gestión de usuarios y tareas.....	34
Usuarios y sus derechos.....	35
Cuadros de Sistemas de Cómputo Voluntario.....	36
1er. Cuadro Comparativo de Sistemas de Cómputo Voluntario.....	36
2do. Cuadro Comparativo de Sistemas de Cómputo Voluntario.....	37
Proyectos que estén utilizando esta metodología.....	38
GIMPS.....	38
Seti.....	38
WCG:Computing for Clean Water.....	38
The Clean Energy Project.....	39
Ibercivis.....	39
LHC@home.....	39
PS3Grid.net.....	40
EDGeS@Home.....	40
El proyecto BOINC en números.....	42
Tabla Comparativa de Estadísticas por Proyecto.....	42
Algunas Estadísticas Representativas.....	43
Fortalezas y Debilidades de estos sistemas.....	45
Fortalezas de BOINC.....	45
Buena Publicidad.....	45
Código Abierto	45
Random Exponential Backoff.....	45
Soporte para gran volumen de datos.....	45
Soporta clientes de múltiples plataformas.....	45
Interfaz Web para voluntarios.....	46
Carga de trabajo configurable.....	46
Seguridad de BOINC.....	46
Principales debilidades de BOINC.....	48
Seguridad de BOINC.....	48
Mala documentación.....	48
Escalabilidad	48
Tareas poco automatizadas.....	48
API de BOINC.....	49
Restricción de usar MySQL.....	49
Fortalezas de Xgrid.....	50
Permisos Unix	50
Ejecución controlada vía sandbox.....	50
Debilidades de Xgrid.....	50
La cantidad de recursos no puede definirse por adelantado.....	50
Las tareas concurrentes no pueden correr en un mismo agente.....	50
La autenticación no es confiable.....	51
Problemas si el controlador se desconecta de la red.....	51
Fuga de memoria en el controlador.....	51
Mejoras a los Sistemas de Cómputo Voluntario.....	52
Arquitectura.....	52
Seguridad.....	55

Uso de firmas digitales y checksums.....	55
Ofuscación.....	55
El uso de redundancia.....	56
Votación por mayoría.....	56
Controles al azar.....	56
Tolerancia a Fallas basada en Credibilidad.....	57
Mejoras al Sistema BOINC.....	58
Involucrar a los usuarios.....	58
Automatizar la creación de proyectos.....	58
Extender la API de BOINC.....	58
Mejorar la seguridad e Integridad de los datos de BOINC.....	58
Caracterizar los voluntarios.....	59
Paralelizar trabajos.....	59
Retribuir los trabajos.....	59
Posibilidades de hacer BOINC rentado.....	60
Seguridad.....	61
Sobrecarga al enviar trabajos al coordinador.....	61
Falsificación de la identidad del coordinador.....	61
Robo de resultados.....	61
Envío de resultados erróneos.....	61
Confidencialidad de la aplicación del cliente.....	62
Resumen de cambios requeridos por BOINC.....	62
Análisis económico.....	63
Conclusiones y Trabajo Futuro.....	67
Bibliografía.....	69
Apéndice A.....	71
Configuración básica de un servidor BINC.....	71
Crear el primer proyecto.....	71
Personalización de la aplicación a ejecutar en el servidor.....	73
Agregar una nueva aplicación.....	73
Actualización de una aplicación - lanzará una nueva versión	74
Demo: Nueva aplicación a partir de test.....	74

Resumen

Cada vez más personas tienen una computadora en su casa que permanece encendida la mayor parte del tiempo con una gran capacidad de cómputo ocioso.

Por otra parte existe una gran cantidad de proyectos de investigación tanto en el ámbito académico como en el industrial; que requieren, día a día, de un mayor poder de cómputo y de almacenamiento.

El problema es aún mayor si esta necesidad de cómputo es variable; es decir, no siempre se está procesando o almacenando grandes volúmenes de datos. Por lo tanto comprar más recursos no sería la mejor opción.

El cómputo voluntario permite darle una solución a este problema, donde el recurso “voluntario” ofrece su capacidad de cómputo o de almacenamiento a proyectos que lo necesiten. Sin embargo, hemos identificado algunos problemas que no han sido solucionados del todo, o a los que la computación voluntaria no ofrece ningún tipo de garantía ni soporte; ya sea porque no fue diseñado para solucionarlos o porque no fueron tenidos en cuenta.

En el presente trabajo, se estudiarán las diferentes herramientas actuales para realizar cómputo voluntario, se mostrarán sus diferentes arquitecturas, sus fortalezas y debilidades. Para terminar, se enumerarán mejoras generales a este tipo de sistemas y en particular al más popular de todos: BOINC.-

Abstract

More and more people are buying new and powerful desktop computers and notebooks. These computers are usually left powered on most of the time with several idle resources.

On the other hand, there are a lot of research projects that each day require more and more computation power and storage.

The problem is even worse, this computation power and storage requirements are variable. So, buying more hardware resources is not always the best option.

Volunteer Computing addresses some of these issues at a very low cost. "Volunteers" offer computation power and/or storage that the research project needs. However, after a brief analysis we have found some issues in these kinds of systems that remain unsolved, or partially solved, but unsecured, or without support. Some of them exist because the entire system was not designed to support that kind of feature, or because it just were not a real concern at that moment.

In this work, we will study the -state of the art- tools used in volunteer computing, the most common architectures, strengths and weaknesses of popular frameworks and finally we will propose some potential improvements to these kinds of systems; and in particular to BOINC: The most popular middleware for volunteer computing.

Introducción

Cómputo Voluntario

La computación voluntaria es un tipo de **computación distribuida** en el cual los dueños de una computadora donan sus recursos de computación (tales como capacidad de almacenaje y de cálculo) a uno o más "proyectos" de investigación. Todos los proyectos comparten una característica en común, necesitan mucho poder de computo o de almacenamiento; al distribuir esa carga entre varias maquinas, se logran los mismos resultados (o aún mejores) que se obtendrían utilizando una súper computadora dedicada únicamente a tal fin.

Hoy, la computación voluntaria está siendo usada en Biología, estudio del clima, análisis de epidemias, Física y mucho más. Algunos ejemplos de este tipo de computación incluyen "GIMPS", "Distributed.net", "Seti@home" y "Folding@home" como veremos mas adelante.

Alrededor del mundo, un millón de computadores participan en computación voluntaria (*volunteer computing*) entregando 1.5 petaflops de poder computacional a los científicos.

Cómputo Voluntario Vs. Cómputo en red (*Grid Computing*)

Existe un concepto generalmente asociado a la computación voluntaria pero en realidad diferente, la computación en Red o "*Grid Computing*". Esta última, a diferencia del primero, permite a las organizaciones tales como empresas, laboratorios de investigación y Universidades compartir recursos computacionales. *Globus* y *Condor* son conocidos por usar, desarrollar y promover software *Grid*.

Globus se utiliza para la investigación y el desarrollo de tecnologías fundamentales para Grids, con la activa participación de la empresa IBM, cuya intención principal es crear una plataforma completa donde compartir aplicaciones y recursos informáticos en Internet.

Condor es utilizado por científicos experimentales cuyo progreso en la ciencia y calidad en investigación esta fuertemente ligada al "*throughput*" computacional que puedan conseguir. Por ejemplo, a muchos científicos le preocupa cuantas operaciones punto flotante por mes o por año pueden conseguir, mas que las que puedan conseguir por segundo o minuto.

Ambas formas de trabajo son parte de una rama de computación más grande llamada "**Computación distribuida**". Sin embargo, son se diferencian en aspectos. Veamos algunos de los más importantes:

1.La computación voluntaria debe acercar a los usuarios poco experimentados.

Los recursos voluntarios pertenecen y son manejados por gente común, no por profesionales IT (Tecnologías de Información). Por tanto, el software debe ser simple de instalar, no puede pedir versiones o configuraciones específicas de sistemas operativos, y debe auto-repararse.

2.La computación voluntaria debe tolerar a los *hackers*.

Los voluntarios son anónimos, y aquellos que tienen un mal comportamiento, quizás por devolver resultados computacionales incorrectos o que piden créditos por trabajo que no han realizado, no pueden ser despedidos ni acusados. El software de computación voluntaria

debe acomodarse a la posibilidad de tales malos comportamientos, por ejemplo, ejecutar cada unidad de trabajo en dos computadores y comparar los resultados.

3. La computación voluntaria “tira” pero no “empuja”.

Los recursos de los voluntarios están detrás de *firewalls* que a veces no permiten las conexiones de red que van llegando. Esto requiere un modelo “*Pull*” en la cual el PC solicita periódicamente unidades de trabajo a un servidor central, en vez del modelo “*Push*” que se usa en la mayoría del software en Red “*Grid*”.

4. La computación voluntaria es un asunto unilateral.

La Computación *Grid* es generalmente simétrica: una organización puede solicitar recursos un día y proveerlos después, la computación voluntaria, al contrario, es asimétrica: los voluntarios proveen recursos computacionales a los proyectos, y no de vuelta.

5. La computación voluntaria demanda grandes relaciones públicas.

La computación voluntaria recae en los voluntarios. Así que los científicos que quieren tener acceso a la computación voluntaria no pueden “comprar la voluntariedad de la gente”, sino más bien deben persuadir al público a que su investigación vale la pena. Es por esto que la divulgación científica es de gran beneficio.

Así que, aunque se suela confundir la computación voluntaria y la computación en Red (“*Grid*”), estos términos son muy diferentes. Usan diferentes capacidades, diferente software y requieren diferentes habilidades y perspectivas de parte de sus usuarios.

Historia

El primer proyecto de cómputo voluntario comenzó en el año 1995 y fue “*The Great Internet Mersenne Prime Search*”, un proyecto cuyo objetivo es encontrar números primos.

Mas tarde surgió un proyecto que se enfoco en el área de la criptografía llamado “*distributed.net*”. Luego otros proyectos fueron apareciendo, entre los que se destaca *Bayanihan* un sistema basado en la web e implementado en lenguaje Java. El término computación voluntaria fue acuñado por su desarrollador Luis F. G. Sarmenta, para referirse a todo tipo de computación distribuida en el que voluntarios donan recursos computacionales para alcanzar los objetivos de un proyecto.

Cabe destacar que el software cliente de los primeros proyectos de computación voluntaria consistía en un simple programa que combinaba la computación distribuida y la infraestructura del sistema, es decir utilizaba una arquitectura monolítica para nada flexible, hasta el punto en que era difícil actualizar sus versiones sin tener que modificar la infraestructura completa.

En 1999 fueron lanzados dos proyectos muy populares, SETI@home y Folding@home. El primero es un proyecto con el ambicioso objetivo de encontrar inteligencia extraterrestre como sus siglas lo indican “*Search for Extra-Terrestrial Intelligence*”. El segundo es un proyecto cuyo fin es investigar acerca de las proteínas, su formación, y el papel que tienen en muchas enfermedades alrededor del mundo.

Tecnologías y Frameworks en Cómputo Voluntario

En la actualidad existen varios *frameworks* que permiten realizar cómputo voluntario. Si bien los siguientes no son todos ellos, han sido elegidos por ser los más populares, por lo que creemos constituyen una muestra significativa:

<p><i>The Berkeley Open Infrastructure for Network Computing</i> (BOINC). Es el middleware mas utilizado para computación voluntaria. De código abierto (LGPL) y desarrollado por un proyecto creado por la NSF en el laboratorio de ciencias espaciales de Berkeley, ofrece software cliente en <i>Windows</i>, <i>Mac OS X</i>, <i>Linux</i>, y otras variantes de <i>Unix</i>.</p>	
<p><i>XtremWeb</i> es un software libre para construir red de computadoras de escritorio recolectando los recursos que no son utilizados en las computadoras de escritorio de los voluntarios (CPU, disco, red).</p>	
<p><i>XtremWeb-CH</i> (XWCH) es un <i>middleware</i> de computación voluntaria que puede desplegar y ejecutar aplicaciones paralelas y distribuidas creando una infraestructura informática con los recursos públicos compartidos. Es descendiente de <i>XtremWeb</i> pero ofrece, como característica distintiva, el soporte para aplicaciones p2p.</p>	
<p><i>XtreamWeb for High Energy Physics</i> (XWHEP) es otro sucesor de <i>XtreamWeb</i>. Es de propósitos generales, bajo licencia GPL, desarrollado por <i>High Energy Physics</i>. Permite correr aplicaciones distribuidas tanto p2p como convencionales (a las que ellos llaman “<i>global applications</i>”).</p>	<p>XWHEP</p>
<p><i>Xgrid</i> es desarrollado por <i>Apple</i>. Su cliente y servidor corren solamente en <i>Mac OS X</i>. Existe un cliente escrito en java para soporte multi-plataforma pero no esta oficialmente aceptado.</p>	
<p><i>Grid MP</i> es un <i>middleware</i> comercial desarrollado por <i>United Devices</i>. Provee un planificador de tareas por prioridad, la posibilidad de excluir aplicaciones (para que no se ejecuten en ciertos nodos), y puede ser usado para manejar dispositivos (desde <i>PCs</i> hasta nodos dedicados en <i>cluster</i>). Ha sido utilizado en varios proyectos de computación voluntaria como <i>grid.org</i>, <i>World Community Grid</i>, <i>Cell Computing</i>, and <i>Hikari Grid</i>.</p>	
<p><i>Simple Light-weight Infrastructure for Network Computing</i> (SLINC) es un <i>framework</i> diseñado para combatir las dificultades que hay con los actuales <i>frameworks</i>. SLINC es flexible y extensible, habilitando a los investigadores a construir proyectos de computación voluntaria de manera mas simple.</p>	<p>SLINC</p>

En la próxima sección, se estudiará BOINC¹, como es su arquitectura, su funcionamiento y composición interna. Dado que el sistema BOINC es por lejos, el más popular de ellos; por su gran cantidad de proyectos corriendo, le daremos un trato especial en este trabajo, analizándolo con más detenimiento que al resto.

Luego se estudiarán las arquitecturas y los componentes de otros sistemas de cómputo voluntario, que si bien son utilizados en menor medida, aportan algunas características propias, que creemos son importantes destacar. Los frameworks elegidos son XstreamWeb, XtremWeb-CH, Xgrid, Grid MP, SLINC y XWHEP.

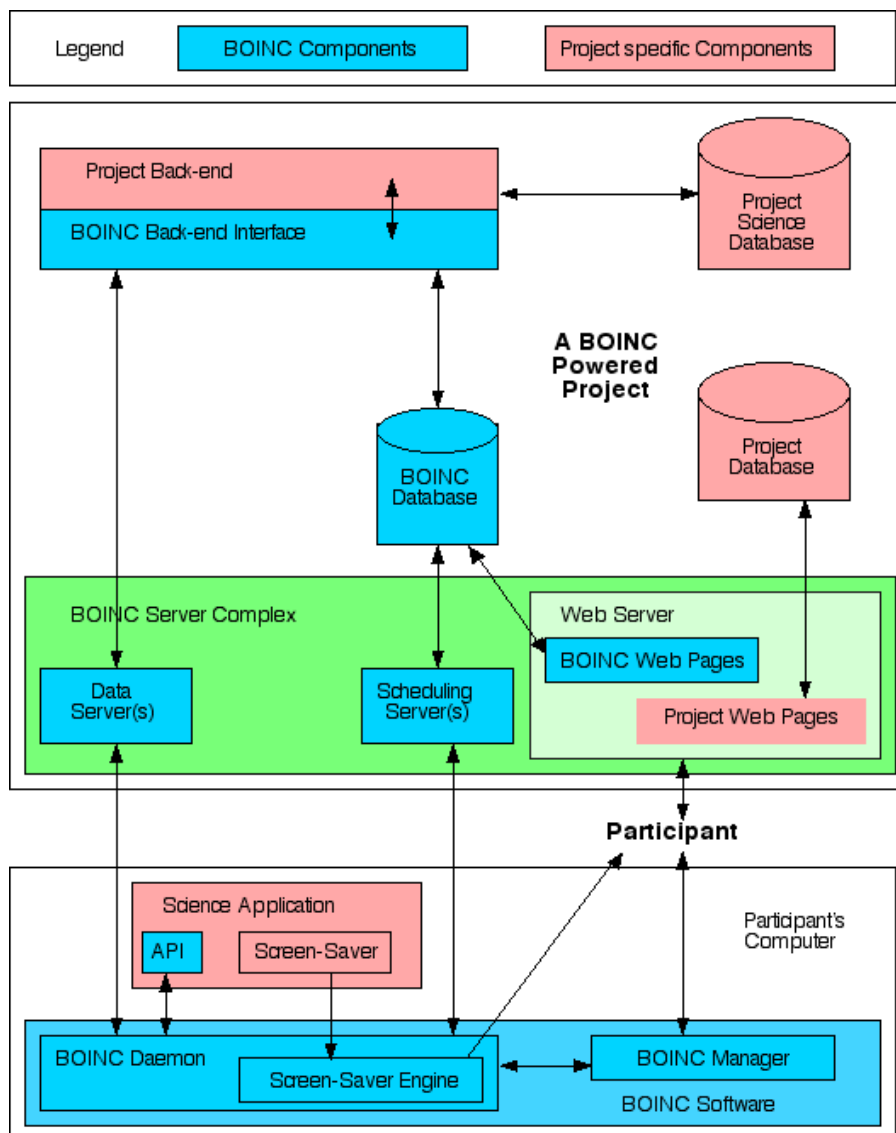
¹ <http://boinc.berkeley.edu/>

BOINC

BOINC es el sistema *middleware* más aceptado y ampliamente utilizado para computación voluntaria por la comunidad científica. Es de código abierto (LGPL) y es desarrollado por un proyecto de investigación financiado por la NSF². El proyecto esta siendo llevado a cabo en el laboratorio de ciencias del espacio de la universidad de Berkeley.

Arquitectura

La arquitectura de BOINC es la madre de todas las arquitecturas para computo distribuido, es la más simple y ampliamente utilizada. El *middleware* se compone de dos grandes bloques lo que se ejecuta en el cliente (voluntario) y lo que se ejecuta en el servidor (donde se aloja y ejecuta el proyecto de investigación en particular).



² La “National Science Foundation” es una agencia federal estadounidense independiente, que promueve el progreso de la ciencia. Actualmente maneja un presupuesto anual aproximado de 7 billones de dolares. <http://nsf.gov/>

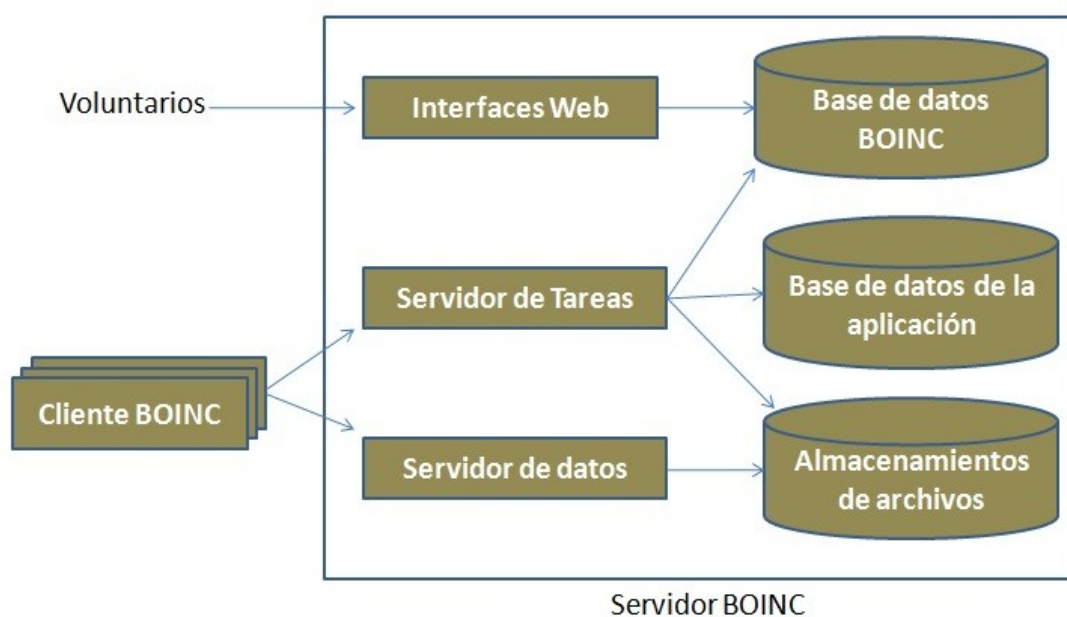
Lo que se ejecuta en el cliente

- El gestor BOINC (BOINC Manager): Este software permite conectarse y controlar el *Boinc daemon*. Permitiendo arrancar y detener el trabajo. Así como gestionar los proyectos a los que se encuentra suscrito el usuario, observar información sobre los volúmenes de información enviada y recibida, progreso de las tareas etc.
- El *daemon* BOINC: es un proceso que se encuentra corriendo constantemente en los ordenadores de los voluntarios. Este demonio se encarga del control de la ejecución de las aplicaciones científicas planificando su ejecución utilizando un organizador de ejecución y tomando las tareas de un *buffer*. Además se encarga de las comunicaciones con el servidor. Este demonio puede ser controlado desde el gestor de BOINC.
- Aplicaciones científicas: son aplicaciones específicas del proyecto en el que se esta participando y que apuntan a resolver un problema particular.

Cabe destacar que el proyecto ofrece el software del cliente para diferentes arquitecturas y sistemas operativos tales como Windows, Mac OS X, Linux, y otras variantes de Unix.

Lo que se ejecuta en el servidor

Cada proyecto es autónomo y utiliza un servidor con los siguientes componentes básicos:

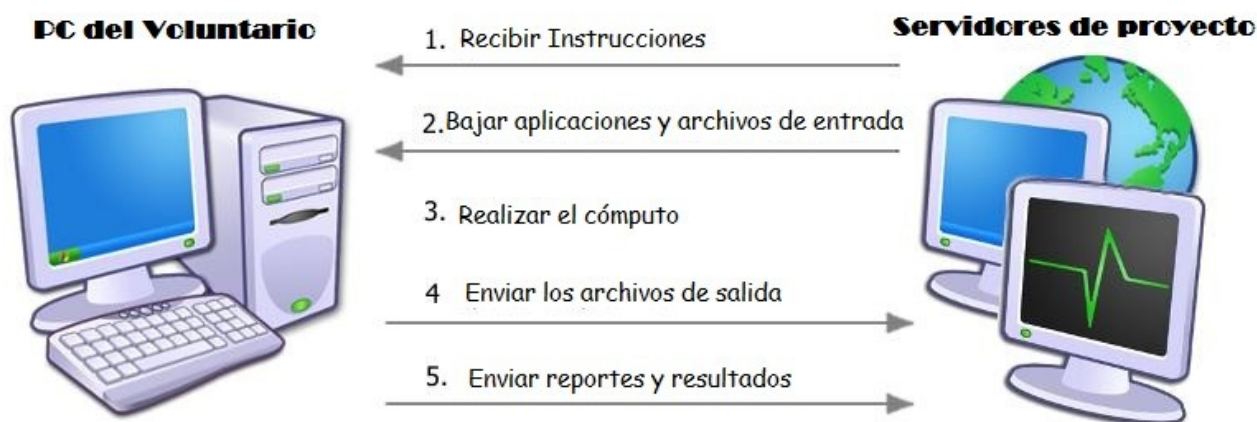


Interfaces web para gestión de cuentas y grupos, pizarras de mensajes y otras funcionalidades.

Un servidor de tareas que crea tareas, las despacha y recibe sus resultados.

Un servidor de datos provee los datos a las computadoras participantes en la forma de Unidades de Trabajo (*Work Units*) para que sean procesados y devueltos al servidor de datos como un resultado.

Funcionamiento de BOINC

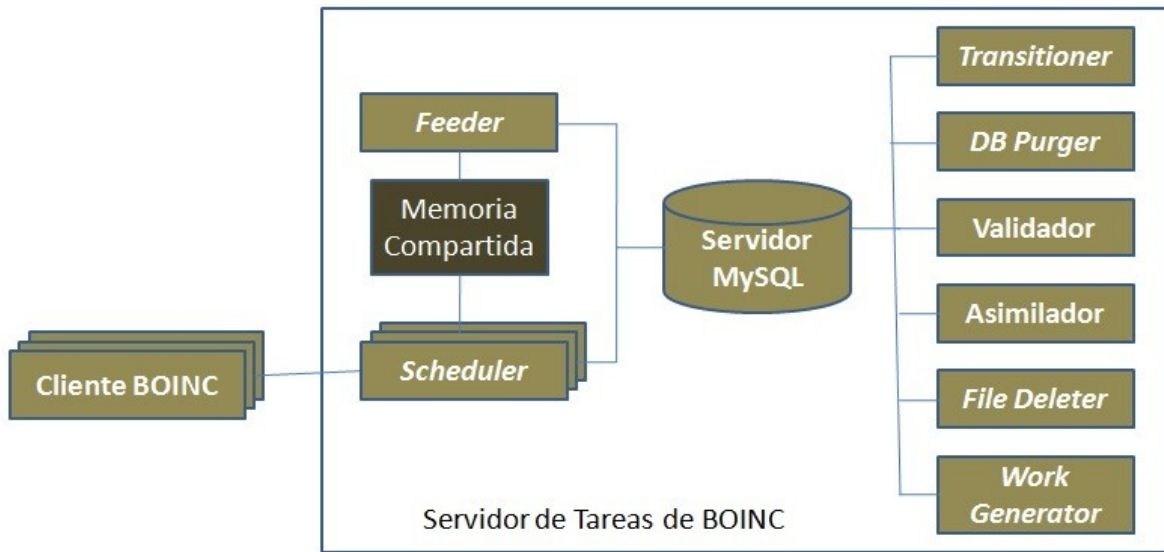


- 1) La PC recibe un conjunto de tareas del servidor de tareas del proyecto. Estas tareas dependen de la PC, es decir, el servidor no va a dar tareas que superen la RAM que tenemos.
- 2) La PC baja el programa a ejecutar y los datos de entrada del servidor de datos. Si el servidor tiene una nueva versión de la aplicación, le va a mandar el ejecutable automáticamente.
- 3) La PC ejecuta la aplicación con esos datos de entrada y produce el resultado.
- 4) La PC sube el archivo de resultados al servidor de datos.
- 5) La PC reporta que completó la tarea al servidor de tareas y recibe nuevos trabajos.

Datos interesantes a tener en cuenta

- Las aplicaciones a ser ejecutadas en los clientes deben ser compiladas para las diferentes arquitecturas en las que el proyecto espera que corran, generando diferentes ejecutables (versiones).
- Las unidades de trabajo, son tareas a realizar, consisten de un conjunto de datos de entrada, un programa a ejecutar y una lista de requerimientos que caracterizan a la tarea (cómputo, número para alcanzar el *quorum*, cantidad de tareas a lanzar, memoria, almacenamiento y un tiempo máximo de ejecución).
- Los resultados de cada unidad de trabajo son empaquetados en archivos de salida que luego son enviados al servidor de datos. La arquitectura de BOINC permite que los servidores de datos puedan estar ubicados de forma distribuida, ya que son simples servidores web y no necesitan acceder a la base de datos de BOINC. Los proyectos de BOINC que usan archivos grandes, utilizan servidores de datos replicados y distribuidos, ubicados en instituciones asociadas.
- Debido a que no es posible confiar en los resultados devueltos por los clientes, es necesario realizar validación de datos, y en caso de ser necesario, reintentos. BOINC provee un mecanismo llamado computación redundante persistente que permite cumplir con ambos requerimientos. Este mecanismo consiste en realizar cada tarea de manera independiente una o más veces y comparar los resultados de salida hasta lograr un “quorum” de salidas equivalentes y generar nuevas instancias de la tarea hasta alcanzar el quorum.

Composición Interna del Servidor de Tareas



El **Servidor de tareas** genera unidades de trabajo, que el *scheduler* reparte a cada cliente según los requerimientos de la tarea y las características del cliente. Estos últimos son quienes piden al *scheduler* las tareas a realizar.

El **Feeder** almacena en cache información de la base de datos que el *scheduler* asigna a algún cliente.

El **Transitioner** verifica y cambia el estado de las tareas. Dependiendo de la situación puede generar nuevas instancias o marcar tareas con errores permanentes o disparar una validación.

El **Validador** compara instancias de un mismo trabajo y selecciona una instancia representativa de la salida correcta. Determina los créditos y se los asigna a un host y un cliente marcando a la tarea como correcta y lista.

El **Asimilador** maneja las tareas que están terminadas ya sea por alcanzar el *quorum* o por error permanente. Manejar las tareas terminas puede implicar guardar los datos en la base de datos o guardar el archivo de resultados.

El **DB Purger** de la base de datos libera las tareas e instancias de tareas que ya no se necesitan y registra el evento en el *log*.

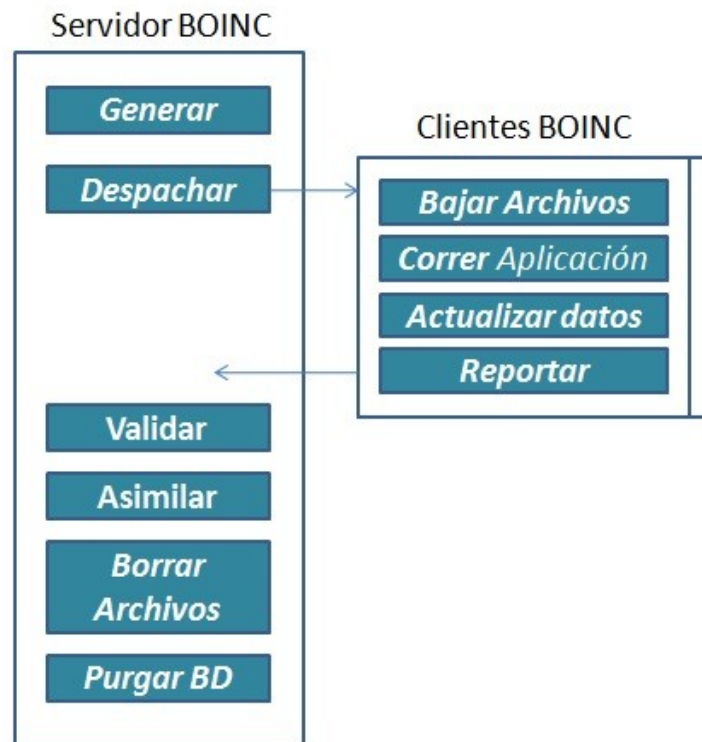
El **file deleter** es el encargado de borrar los archivos de entrada y salida cuando el trabajo se haya terminado de computar.

El **work generator** es el encargado de generar los trabajos junto con los archivos de entrada necesarios.

La comunicación entre ellos es canalizada a través de la base de datos de BOINC creando entradas o modificando *flags* para que otro programa realice alguna acción. Por ello los programas deben estar chequeando la base de datos de manera periódica para atender esos registros y limpiar *flags*.

Ciclo de vida de un trabajo

BOINC esta preparado para manejar millones de host voluntarios y millones de trabajos por día. Para maximizar la performance de nuestros proyectos, es necesario entender el ciclo de vida de un trabajo:

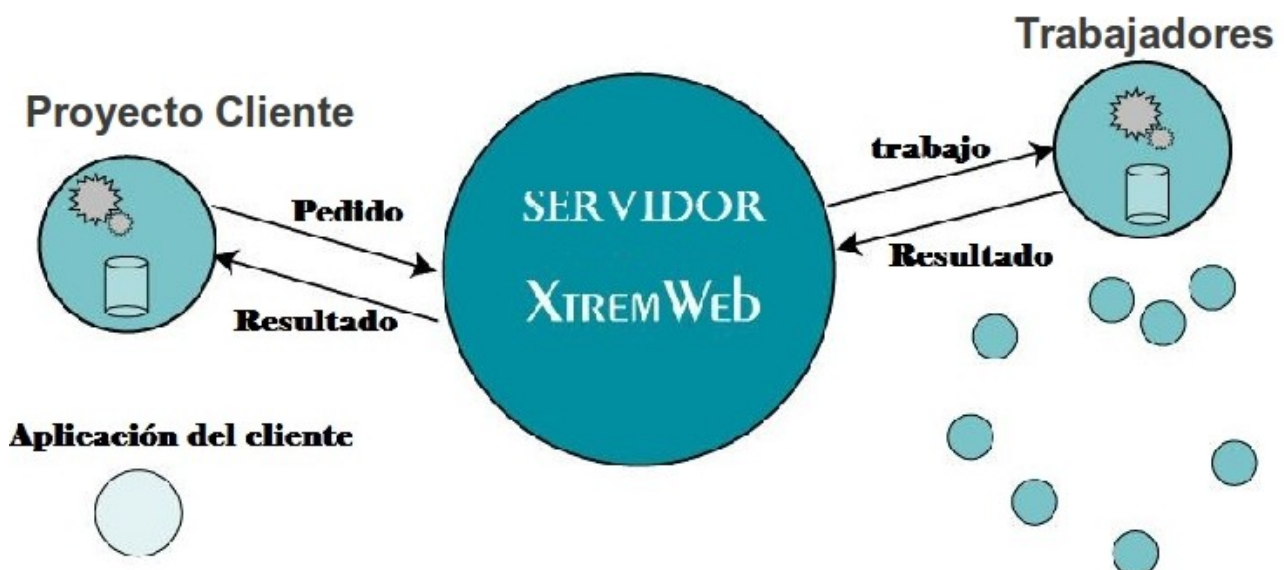


- El *work generator* (provisto por el cliente) crea el trabajo y sus archivos de entrada asociados.
- BOINC crea una o más instancias del trabajo.
- BOINC envía las instancias a los diferentes *hosts*.
- Después de terminar los trabajos que se encuentran en progreso, el trabajo es desencolado y el *host* ejecuta el trabajo y luego sube los archivos de salida.
- El host reporta que se completo el trabajo, posiblemente, después de esperar cierto tiempo para reducir los pedidos al *scheduler*.
- Un validador (provisto por el proyecto de investigación) chequea los archivos de salida, quizás comparando réplicas.
- Cuando una instancia valida es encontrada, el *file deleter* borra los archivos de entrada y salida.
- Se ejecuta un programa para purgar la base de datos (*DB Purger*), que borra las entradas del trabajo y sus instancias.

XtremWeb

XtremWeb es un software libre para construir red de computadoras de escritorio recolectando los recursos no utilizados de las computadoras de escritorio (CPU, almacenaje, red). Sus principales características permiten múltiples usuarios y múltiples aplicaciones. XtremWeb convierte un sistema de LAN o Internet con recursos distribuidos en un ambiente que ejecute aplicaciones altamente paralelas. Día a día, XtremWeb proporciona poder de computación a los científicos de funcionar con su código en la Física de alta energía, BioMolecular, Matemáticas e innumerables dominios científicos y/o industriales. XtremWeb es altamente programable y adaptable una amplia gama de los usos, de requisitos computacionales (*data/CPU/network-intensive*) y de infraestructuras de cómputo (*clusters*, PC de escritorio, *multi-Lan*) de manera manejable, escalable y segura. XtremWeb es también una investigación de redes Grid y tecnologías P2P.

Arquitectura



La arquitectura general de XtremWeb se gráfica en la figura anterior: un servidor organiza los cálculos sobre máquinas distantes anónimas llamadas trabajadores (*workers*). A petición de un cliente que desea ejecutar su aplicación, el servidor XW lo deriva a un trabajador transmitiéndole su código y sus datos de entrada. A la recepción de los resultados, el servidor XW los devuelve al cliente.

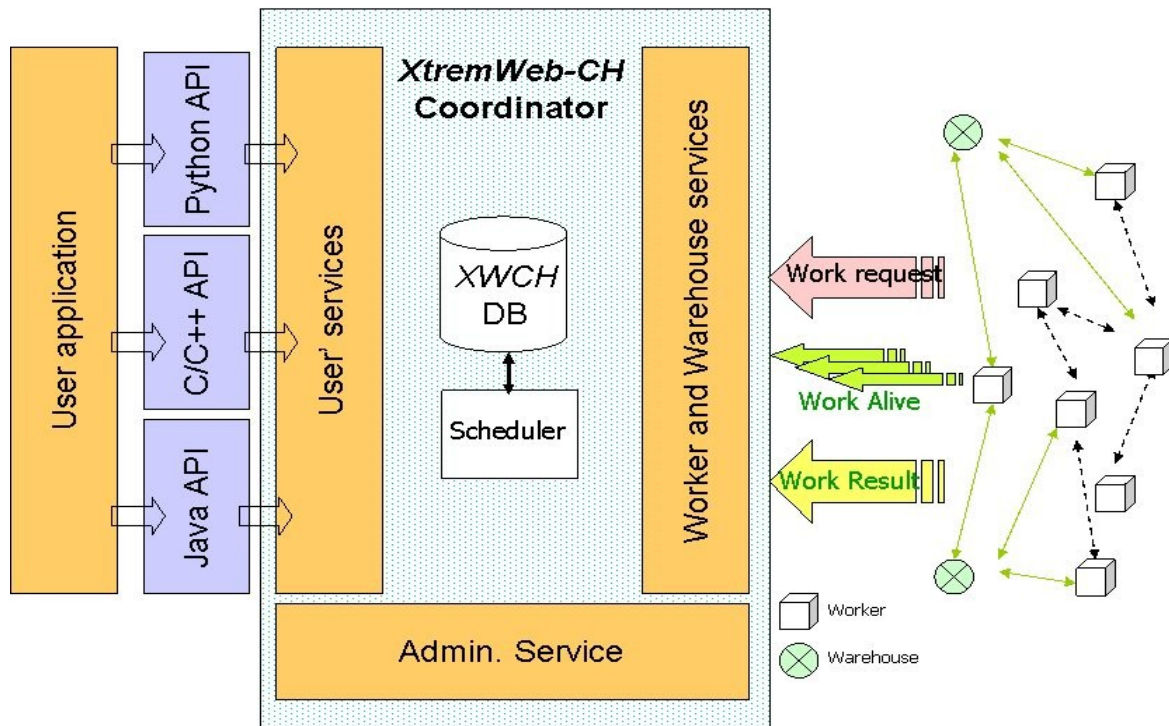
XtremWeb-CH

En su versión original, XtremWeb (de ahora en mas, “XW”) soportaba aplicaciones mono-módulo. XtremWeb-CH es una versión mejorada de XW. Este medio ambiente permite la ejecución de aplicaciones paralelas/distribuidas compuestas de módulos comunicados.

Las aplicaciones son descritas por un gráfico flujo de datos, donde los nodos son los módulos de programa y los vínculos entre nodos representan los intercambios de datos entre estos módulos. El gráfico flujo de datos está representado por un archivo XML que describe los módulos y los datos intercambiados. Sólo se destina un módulo a un trabajador si sus datos de entradas están disponibles. Se dice que un dato está disponible si fue generado por un módulo anterior. Inicialmente, el único dato de entrada disponible es el que es proporcionado por el usuario para lanzar el (o los) primero(s) modulo(s).

Los distintos módulos de la aplicación (a excepción del primero) se bloquean inicialmente: no pueden ser ejecutados puesto que sus datos de entrada no están disponibles. Al final de la ejecución de un módulo por un trabajador, éste informa al servidor. Un proceso particular de XW-CH, llamado espía, recorre entonces todos los módulos bloqueados para liberar aquel cuyos datos de entrada están ahora disponibles. Encontramos particularmente eficiente, y por ello lo destacamos, que la comunicación entre los módulos de XtreamWeb-CH se haga directamente entre los nodos (*workers*) sin tener que pasar por el servidor. Esta nueva funcionalidad es una extensión de XW-CH con relación a XW. En efecto, en XW las comunicaciones entre trabajador no pueden tener lugar sino mediante el servidor. Este último permite entonces reducir la carga de comunicación al servidor. XW-CH se acerca aún más pues al concepto P2P: los nodos tienen las mismas funciones, el mismo poder. La arquitectura de XW-CH parcialmente se descentraliza con relación a la de XW. Cuando un trabajador termina su ejecución, almacena sus resultados en un fichero temporal y envía una señal al servidor XW-CH que le indica el final normal de su ejecución y el sitio del fichero resultado: dirección IP y directorio. Cuando el servidor XW-CH destina los módulos recientemente liberados a trabajadores libres, les envía también el sitio de su dato de entrada. La transferencia de los datos se hace directamente pues entre trabajadores. Cuando uno de los trabajadores se protege (dirección interna, cortafuego, etc), la comunicación directa no puede tener lugar. El intercambio de datos se hace entonces mediante una máquina enlace.

Arquitectura



Componentes

XWCH consiste en tres componentes principales: el coordinador, el trabajador, y el almacén. Actualmente, no hay paquetes dedicados del cliente (quien solicita un trabajo). En su lugar, los trabajos o las solicitudes se presentan usando la API o páginas Web.

El coordinador

El coordinador acepta las peticiones de la ejecución que vienen de clientes, asigna las tareas a los trabajadores según una política de previsión y la disponibilidad de datos, transfiere códigos binarios a los trabajadores (en caso de necesidad), supervisa la ejecución de la tarea en trabajadores, detecta caídas del trabajador o su desconexión y relanza tareas en cualquier otro trabajador disponible.

El trabajador

El trabajador extrae la tarea asignada, comienza el cómputo y luego de finalizada la tarea devuelve los datos. Los trabajadores funcionan en modo “*pulling*”, es decir, son los que “tiran” de la información, al iniciar la conexión con el coordinador para conseguir trabajos o los datos de entrada (*Work Request signal*), o para notificar al coordinador sobre el estado del trabajo (*Work Alive signal*). Los trabajadores pueden, por lo tanto funcionar detrás de un *router*.

El almacén

Los almacenes son utilizados por los trabajadores para transferir los datos de entrada necesarios

para ejecutar. Otros trabajadores toman sus datos requeridos por la tarea. Un nodo del almacén actúa como un depósito o servidor de I/O de archivos para las tareas.

El ciclo de trabajo de XtremWeb-CH es el siguiente:

1. El trabajador recibe una lista de los almacenes disponibles cuando se registra con un coordinador cercano (Solicitud de Registro).
2. Cuando un trabajador envía una solicitud de trabajo para ejecutar una nueva tarea, recibe como respuesta, el código binario de la tarea asignada y los dos lugares de sus datos de su entrada.
3. Cuando un trabajador termina la ejecución de una tarea, carga su resultado en uno de los almacenes conocidos (seleccionado al azar). Por lo tanto, el resultado se almacena en el trabajador y en el almacén.
4. El trabajador envía el resultado del trabajo al coordinador, con las dos ubicaciones de los resultados producidos y se da por concluido el trabajo.

Xgrid

Xgrid es un programa de software propietario y un protocolo de computación distribuida desarrollado por la subdivisión de la Computación Avanzada Grupo de Apple. Este permite coordinar ordenadores en red para contribuir a una tarea común.

Proporciona a los administradores de redes un método simple para la creación de un *cluster*, lo que les permite explotar el poder de cómputo no utilizado. Es ideal para los cálculos que se pueden dividir fácilmente en operaciones más pequeñas.

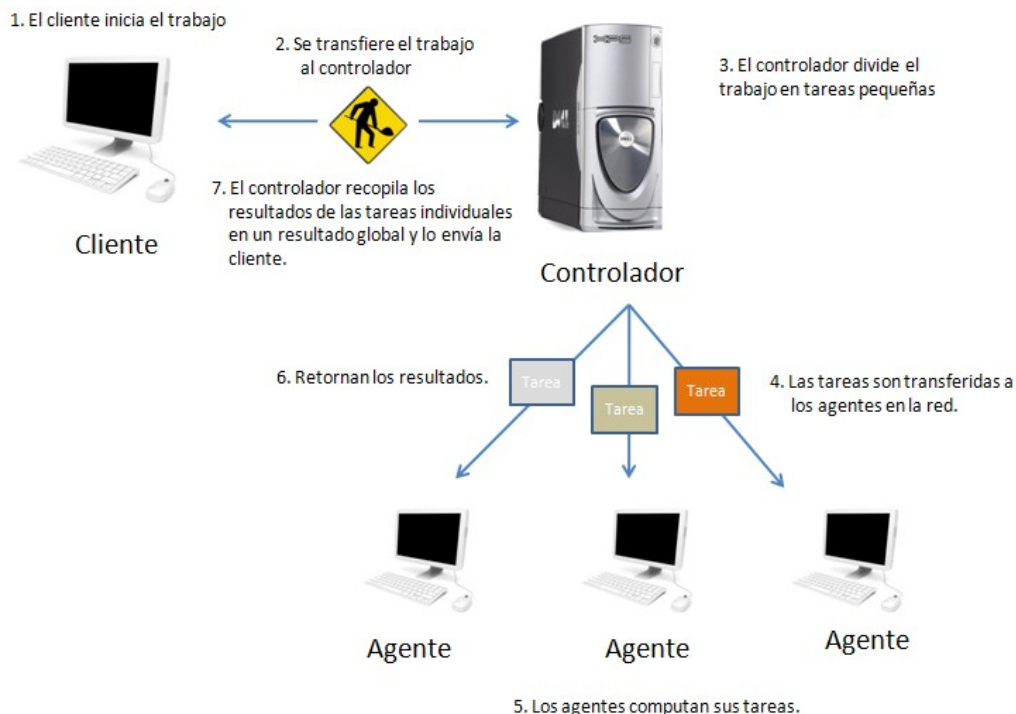
Como cliente, Xgrid está pre-instalado en todos los ordenadores con *Mac OS X 10.4* o posterior.

El controlador Xgrid, el planificador de tareas de la operación Xgrid, también se incluyen dentro del *Mac OS X Server* y como una descarga gratuita de *Apple*.

Apple ha mantenido línea de comandos como un mecanismo minimalista para el control de trabajos, mientras que a la vez que proporciona una API para desarrollar herramientas más sofisticadas en torno a ella.

Componentes

Una Xgrid se divide en tres componentes lógicos: Agentes, Controladores y Clientes. Los equipos cliente envían trabajos (un conjunto de tareas) que quieren correr en un equipo controlador. El controlador encola trabajos de los clientes y distribuye las tareas a los equipos agente. Las computadoras agente ejecutan las tareas e informan sus resultados y el estado de vuelta al controlador donde se almacenan hasta que son eliminados por el cliente. A continuación se detalla un gráfico con los componentes de *Xgrid* en funcionamiento y sus relaciones:



Características sobresalientes

1. Los clientes pueden consultar el controlador de forma asíncrona sobre el estado de un trabajo y los resultados.
2. Cualquier equipo OSX puede comportarse como cualquiera de estos componentes.
3. Un *Mac* puede ser más de un componente a la vez; es decir es posible ser agente, regulador y cliente al mismo tiempo.
4. Hay **solo** un controlador por cada grid.
5. Los clientes pueden enviar los trabajos a los controladores de diferentes grids.
6. Los agentes pueden trabajar por más de una grid.

Xgrid tiene una paleta de pequeña de opciones de instalación. Las dos primeras decisiones a tomar son el tipo de autenticación y autorización a usar y si se necesita un sistema de archivos compartidos. Un sistema de archivos compartido al que todos los agentes pueden tener acceso puede ser muy beneficioso para muchos problemas de computación, pero no es apropiado para todas las redes.

Sistema de archivos compartido

En general los ejecutables necesitan acceder a datos y librerías para poder hacer un trabajo. Puede que estos archivos no estén disponibles en el agente y sea necesario traerlos desde la fuente. Es más, puede que las aplicaciones necesiten transferir los resultados generados al *host* administrador. El sistema Xgrid ofrece facilidades muy limitadas para estas funciones, en general se transfieren a todos los nodos, necesiten o no los datos. Para evitar esto y usar solo la información que se necesita y aprovechar mejor los datos transferidos es recomendable utilizar un sistema de archivos compartido en alguno de los otros ordenadores.

Los sistemas de archivo pueden ayudar a relajar muchas limitaciones del sistema, por ejemplo pueden ofrecer un sistema de comunicación económicos entre procesos, persistencia. Y además pueden permitir un sistema de estadísticas mas sofisticados cuando el proyecto utiliza tableros de puntuaciones.

El sistema de transporte incluido sacrifica eficiencia por simplicidad, los archivos de entrada enviados al sistema no son almacenados en cache, aún cuando los datos de entrada son necesarios para diferentes tareas dentro del mismo *host*. La información es retransferida para cada tarea. Los datos binarios son transferidos sin comprimirse codificados en xml ocupando 4/3 más que el archivo original. Y para tareas simples el tráfico de red se duplica pues todo debe ir desde el cliente hasta el controlador y desde el controlador al agente. Lo mismo sucede con los archivos de salida, que son generados por el controlador y enviados posteriormente al cliente. Mientras que con un sistema de archivos compartido por el cliente, el agente simplemente toma y coloca los datos directamente donde se necesiten.

Grid MP

Grid MP es un paquete de software de computación distribuida pero de licencia comercial. Es desarrollado, soportado y vendido por la empresa Univa (*ex United Devices*), la que también desarrolla soluciones para *clustering* (llamada *UniCluster*) y *cloud computing* (llamada *UniCloud*). El último *release* estable es de Julio de 2008, soportando la gran mayoría de los sistemas operativos actuales: Linux, Windows, Mac OS X, Solaris, etc.

Grid MP es una pieza relativamente compleja de software, compuesta principalmente a base de código C++ con un significativo uso de la STL (Standard Template Library). Perl es muy utilizado para los scripts de pruebas automatizadas y de construcción. Su SDK proporciona ejemplos escritos en Perl, C++, Java y C#.

Su código también depende en gran medida de varios cientos de procedimientos almacenados de SQL que se han implementado para DB2 y Oracle. Todos los procedimientos SQL y las definiciones de esquemas se mantienen en XML (llamado internamente "datadef") que le permite generar automáticamente encabezados, definiciones de estructuras, constantes, enumeraciones, creación de tablas, creación de índices, código de procedimientos almacenados, SOAP WSDL, etc. Algunas de las ventajas de este sistema incluyen:

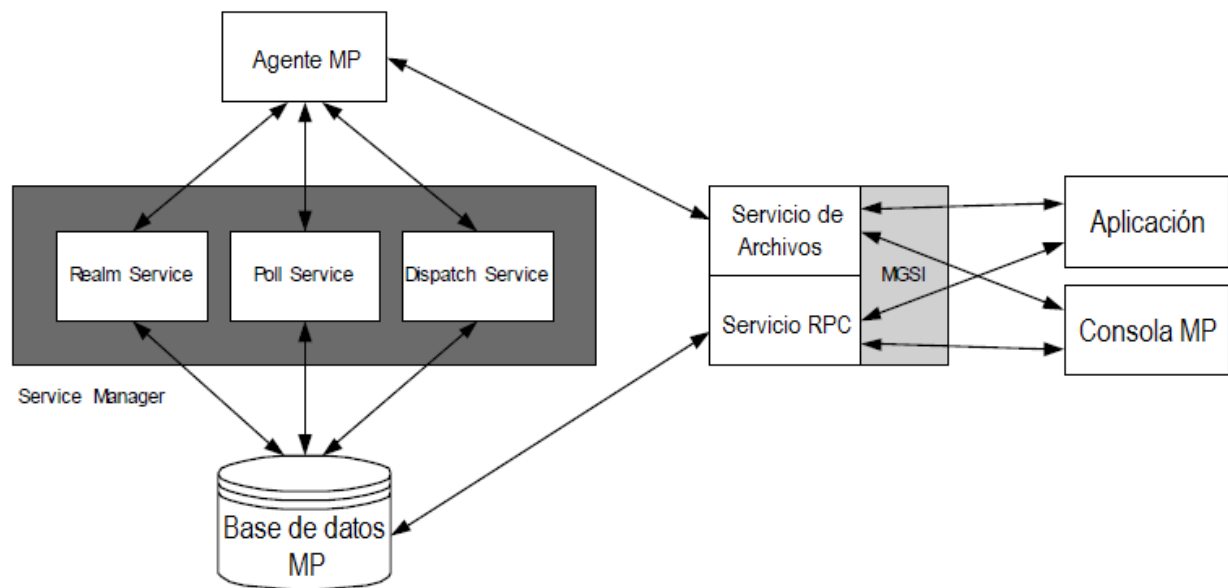
- Agrupamiento de recursos basados en su ubicación y sus capacidades.
- Un ambiente de ejecución seguro.
- Capacidad de encriptar y comprimir automáticamente los datos en la red y durante operaciones de persistencia.
- *Scheduling* basado en la prioridad de los trabajos.
- Restricciones de recursos.
- Reasignación automática de trabajo en caso de falla.
- Asignación de trabajos basado en afinidad por localidad de datos en cache.
- Checkpoints en aplicaciones para mejorar la distribución de carga.

Seguridad

Algunos de los aspectos más importantes respecto de la seguridad que se tienen en cuenta en este *framework* son:

- Seguridad en el agente: Creación de una cuenta de usuario propia con privilegios mínimos para ejecutar las aplicaciones, encriptación de información de cuentas y de información de inicialización en tiempo de instalación, etc.
- Firmas digitales: Utiliza el mecanismo de firmas digitales para validar módulos de programas antes de ejecutarlos.
- Seguridad de los servicios: Uso de *token keys*, control de nombres de usuario y grupo para controlar el acceso a los servicios.

Arquitectura



Cada nodo o dispositivo, en la red ejecuta un agente de MP. Un servidor central mantiene el estado global del sistema distribuido, controlando el funcionamiento de la red. Una de sus responsabilidades es la distribución de tareas para ejecución en cada agente de MP.

Cuando el Agente MP se inicia, en primer lugar, se conecta con el dispositivo en el que se ejecuta el Servicio de Dominio para autenticarse y obtener las credenciales. A continuación, el agente MP se contacta con el Servicio de *Despacho de Tareas* para solicitarle trabajo. A continuación el agente MP, descarga los archivos necesarios desde el Servicio de Archivos y comienza a procesar. Mientras procesa el trabajo, el agente informa estadísticas de uso al servicio de estadísticas. Cuando el trabajo está terminado, el agente MP sube los resultados al Servicio de Archivos, y regresa al servicio de Despacho de Tareas para obtener más trabajo.

El usuario puede enviar trabajos a la plataforma MP con diferentes interfaces de plataforma, por ejemplo, MP Interfaz de Servicios (MGSI) y la consola de gestión.

MGSI es una interfaz de programación basada en la web, desarrollada en XML y SOAP. MGSI contiene funciones para manipular todos los objetos en la plataforma Grid MP mediante el servicio de archivos y/o el servicio RPC.

MGSI se utiliza para escribir los servicios de aplicación, que envuelven la aplicación provista por el usuario (*wrapper*) para la ejecución de las tareas de pre-procesamiento y post-procesamiento.

SDK UD ofrece la posibilidad enviar tareas por lotes, los datos en paralelo (*data parallel*), y el empleo de trabajos basados en MPI.

Componentes de Grid MP

Program Loader

El *program loader* es un ejecutable que debe estar presente en todos los casos. Su función es cargar los ejecutables en la memoria para su ejecución, y proporciona el mecanismo de encriptación y compresión automática para sus ejecutables.

Servicio MGSI - RPC

La “MP Grid Services Interface (MGSI)” está compuesta por el servicio RPC y un servicio de archivo. El Servicio de MGSI-RPC es una interfaz de programación que proporciona, a aplicaciones de terceros, el acceso a la consola MP, a los Servicios de MP y la base de datos. El MGSI también incorpora control de acceso para los usuarios de estos servicios.

Consola MP

La Consola presenta un cliente liviano basado en Web para la gestión de los servicios MP, Agentes MP, Aplicaciones, Trabajos, y Datos.

Base de datos

La base de datos es el principal lugar de almacenamiento de información común a toda la plataforma MP. Aquí se almacena la totalidad del estado del sistema:

La mayoría de las interfaces y servicios se comunican directamente con la base de datos.

Agente MP

El Agente es una aplicación liviana que corre en un dispositivo y gestiona el procesamiento de las tareas. Se comunica con todos los servicios del sistema, recibe *tokens* del *Realm Service* para comunicarse con los otros servicios, pide trabajo al *Dispatch Service*, descarga módulos de programa y datos del *Servicio de Archivos* y otros sitios web, a su vez, terminada la computación; sube sus resultados a través del *Servicio de Archivos*. Además le envía datos al servicio de estadísticas.

Utiliza firma digital para evitar la modificación maliciosa tanto de sus datos y como de las aplicaciones.

Service Manager

Su principal función es asegurarse de que todos los servicios requeridos para el correcto funcionamiento del sistema, estén haciéndolo de manera adecuada.

Este servicio cumple con los siguientes roles:

- ✓ Arrancar y detener todos los demás servicios.
- ✓ Periódicamente verificar que los servicios están levantados y ejecutándose. Si alguno se encuentra detenido o ha fallado, el servicio es relanzado.
- ✓ Cuando los servicios están corriendo en diferentes máquinas, levantar una copia local del *service manager* en cada sistema de manera independiente.
- ✓ En caso de fallar, el mismo se relanza automáticamente.

Realm Service

El *Realm Service* gestiona toda la autenticación de los agentes al sistema y el acceso de estos a los recursos. Los agentes se comunican con este servicio para registrarse y autenticarse.

Este servicio cumple con los siguientes roles:

- ✓ Validación de los agentes existentes, creación de cuentas y registro de dispositivos.
- ✓ Provisión de *tokens* para la autenticación distribuida de dispositivos
- ✓ Sincronización temporal
- ✓ Transmisión de direcciones de red de otros dispositivos

Poll Service

El *Poll Service* comunica al agente los comandos provenientes de otros servicios y recoge periódicamente el estado de los mismos.

El servicio de estadísticas es responsable de:

- ✓ Conglomerar información sobre carga computacional de los agentes.
- ✓ Asistir en el envío de comandos a los agentes, tales como re-autenticarse, reiniciar, abortar, apagar el agente, entre otros.
- ✓ Monitorear el correcto funcionamiento de los agentes.

Dispatch Service

El *Dispatch Service* asigna unidades de trabajo a los dispositivos basándose en sus capacidades y su disponibilidad. Además recibe el estado de resultados de los dispositivos. Los dispositivos ociosos de la plataforma Grid MP se conectan con este servicio para recibir nuevas unidades de trabajo. Este servicio incluye un planificador de unidades de trabajo que tiene en cuenta, prioridad del trabajo, prioridad del usuario, requerimientos de recursos, preferencias del dispositivo y capacidades del dispositivo. La plataforma soporta además planificación y ejecución de tareas redundantes para asegurar la precisión de los resultados.

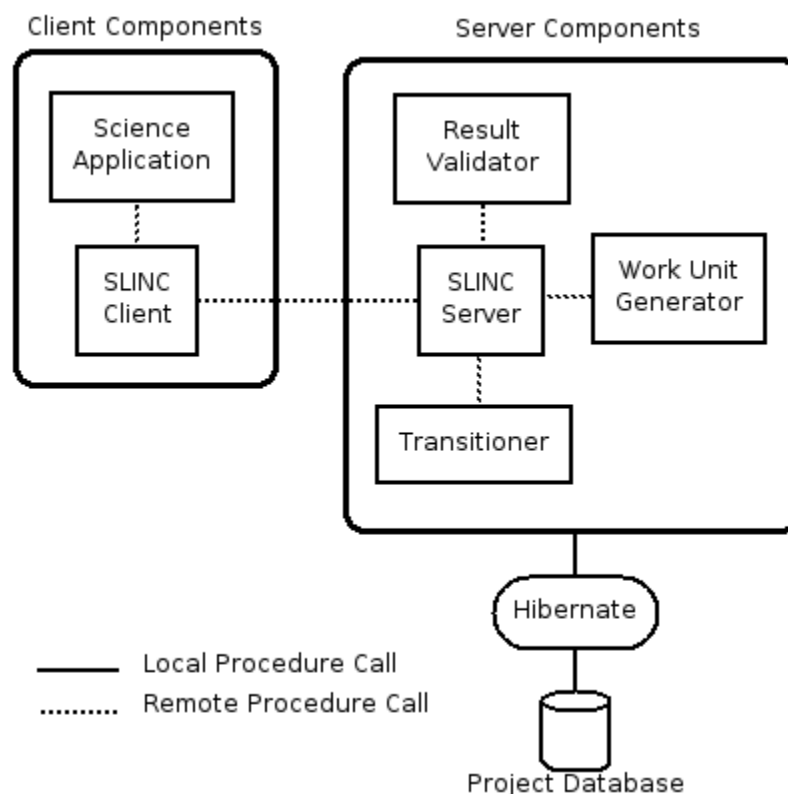
Servicio de Archivos

El servicio de archivos es un servicio seguro de transferencia de archivos (utiliza encriptación SSL). Es utilizado tanto para subir como para bajar archivos de datos desde y hacia los agentes como así también módulos de las aplicaciones. Además provee tareas de mantenimiento en segundo plano para eliminar los archivos que ya no se necesitan.

SLINC

SLINC es un *framework* para combatir las dificultades encontradas en los actuales *frameworks*. SLINC es flexible, extensible, y habilita a los investigadores a construir proyectos de computación voluntaria de manera más simple. Fue desarrollado en un primer momento por James Baldassari de "Network Centric Systems", David Finkel y David Toth del Departamento de Ciencias de la Computación del "Worcester Polytechnic Institute". Diseñado bajo licencia GPL con repositorio en *sourceforge*, ahora es mantenido por la comunidad. Al estar desarrollado en Java, es multi-plataforma e independiente del S.O.

Arquitectura



SLINC consta de varios componentes, y fue diseñado de manera similar e influenciado por BOINC; hasta el punto de referenciarlo en el *paper* donde es presentado el *framework*. Algunos de sus componentes son proporcionados por SLINC (siguiendo la filosofía de BOINC), pero otros necesitan ser desarrollados por los creadores del proyecto de computación voluntaria. Los componentes se dividen en dos: los que están del lado del cliente y los componentes que están del lado del servidor. Todos los componentes son procesos separados que se comunican a través de XML-RPC, una especificación abierta para llamadas a procedimientos remotos basados en XML, queremos subrayar que dicha especificación permite que los módulos se puedan desplegar en diferentes equipos físicos, es decir brinda transparencia con respecto a la ubicación de los recursos. Los componentes del lado del servidor están diseñados para ejecutarse en máquinas gestionadas por los administradores del proyecto, mientras que los componentes cliente están diseñados para ejecutarse en las computadoras de personas que deciden colaborar con el proyecto (los voluntarios). Los componentes del lado del servidor son responsables de mantener la base de datos del proyecto, la partición de datos de entrada en unidades de trabajo, la distribución de las unidades de trabajo a los clientes, y el procesamiento y validación de los resultados de cada unidad de trabajo. Hay una sola aplicación, en el servidor central de aplicaciones, al que todos los clientes deben conectarse. Los componentes del lado del cliente solicitan unidades de trabajo al servidor del proyecto, calculan el resultado para cada unidad de trabajo, y lo devuelven al servidor. Existe un único cliente que se ejecuta en el ordenador de cada voluntario y se comunica con el servidor central del proyecto.

Según la filosofía de sus creadores quisieron hacer de SLINC algo fácil de usar en pequeños proyectos, suficientemente poderoso para soportar grandes proyectos y lo suficientemente flexible para correr en diferentes ambientes.

Para lograr este objetivo, utilizaron la librería *open-source Hibernate*, que sirve como capa de abstracción al acceso a la base de datos. A través de ella, SLINC soporta la mayoría de las bases de datos mas populares, incluyendo MySQL, PostgreSQL, Oracle, y Microsoft SQL Server.

Cabe destacar que SLINC NO utiliza encriptación dado que no fue uno de los objetivos principales de sus autores, pero lo tienen planeado como *feature* futuro.-

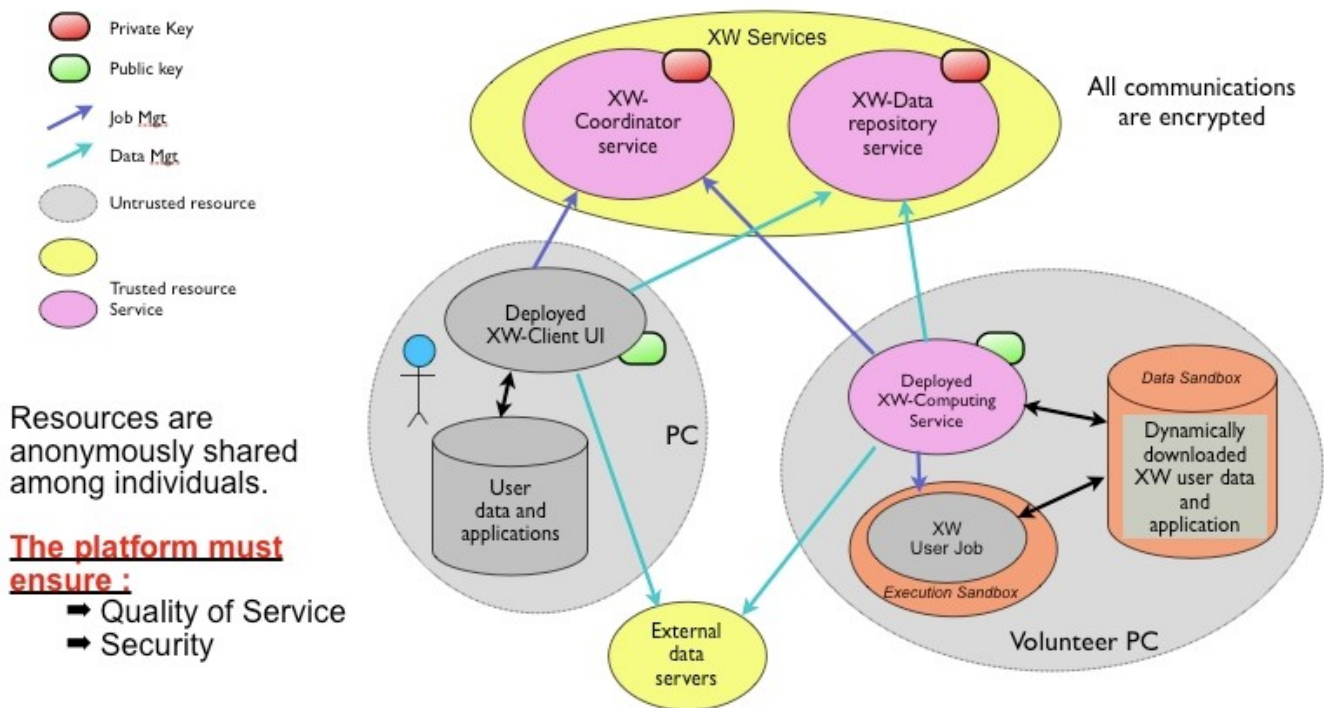
XWHEP

XtremWeb for High Energy Physics (XWHEP) es una plataforma de computación global que apunta a conglomerar recursos computacionales de toda internet. Contrario a lo que el nombre podría hacer pensar, este es un *framework* de uso general, no dedicado específicamente a la comunidad de física de alta energía (HEP). Es *Free Software* (bajo licencia GPL), abierto (Open Source) y sin fines de lucro.

El proyecto se basa en XtremWeb, que fuera iniciado por el laboratorio de investigaciones informáticas ([LRI](http://www.lri.fr/)³). En una iniciativa multi-disciplinaria, el [Laboratoire de l'Accélérateur Linéaire \(LAL\)](http://www.lal.in2p3.fr/)⁴ y el LRI juntaron sus esfuerzos para realizar lo que era una plataforma de investigación cuyo objetivo era el estudio de sistemas distribuidos a gran escala. Los primeros casos de uso naturalmente fueron aplicaciones de del laboratorio de HEP (LAL es un laboratorio de HEP). Este trabajo en conjunto fue un éxito, a tal punto que el LAL continuo, mejoró y expandió la funcionalidad de su antecesor en muchas maneras bajo el nombre de XWHEP.

³ <http://www.lri.fr/>

⁴ <http://www.lal.in2p3.fr/>



Componentes

El proyecto XWHEP esta formado por tres partes:

El servidor es la parte centralizada, encargada de la gestión de la plataforma; mientras que los clientes y los voluntarios son la parte distribuida del sistema y son implantados en las PCs de los usuarios esparcidos por toda Internet. Esta arquitectura corresponde a la mayoría de las actuales plataformas para computo voluntario, donde un servidor centralizado garantiza la coherencia, integridad y seguridad, mientras que las partes distribuidas (el cliente y el trabajador) son responsables de brindar la interfaz de usuario y capacidad computacional respectivamente.

El servidor y las partes distribuidas son conocidas como el *middleware* y cuando se los junta forman una red o una implantación.

El servidor centralizado es la única entidad de confianza, debe ser altamente segura y mantenida por el administrador de la plataforma para asegurar la seguridad y la calidad de servicio de toda la plataforma.

Las partes distribuidas no son de confianza en esencia. El *middleware* es el encargado de garantizar la seguridad y los niveles de calidad de servicio implementando servicios y protocolos.

Una de las principales virtudes ya implementadas en *XtremWeb* es la replicación de servidores, tal como se describe en el paper "*RPC-V: Toward Fault-Tolerant RPC for Internet Connected Desktop Grids with Volatile Nodes.*".

XWHEP extiende el servidor de *XtremWeb* introduciendo la noción de permisos de acceso y "datas" (librerías, ejecutables, archivos de entrada, archivos de salida, etc). El servidor puede actuar como un servidor de datos pero las *datas* pueden ser guardadas en cualquier lugar accesible mediante una URI. La versión actual incluye soporte para *http*, *attic*, e implementa un esquema propio (*xw*).

XWHEP permite a cualquier persona usar una plataforma de computo P2P, participar (proveyendo recursos computacionales a un proyecto de computación global o un proyecto de computo P2P uniéndose a cualquier plataforma XWHEP, fácilmente crear un sistema de cómputo global.

Los participantes descargan el software cliente, éste estará constantemente monitoreando el uso del sistema y cuando el uso del mismo se vuelva insignificante, el trabajador comenzará a ejecutar tareas.

Servicio de Archivos y Seguridad

La plataforma no confía en “nada”, implementa todos los mecanismos requeridos para garantizar la integridad de sus aplicaciones, *datas*, tareas, y los trabajadores conectados.

Los permisos de acceso, la gestión de usuarios y sus permisos aseguran entre todos los niveles de seguridad de manera tal que la gestión de usuarios garantice la seguridad, los grupos de usuarios y los permisos de accesos aseguran la confidencialidad, el servidor garantiza la integridad y la persistencia.

La seguridad es garantizada en tres niveles:

Nodos de computo: aquí se utiliza una caja de arena para aislar la ejecución de las tareas, solo las aplicaciones validadas provenientes del repositorio son candidatas a correr en los nodos.

Integridad de aplicaciones y datos: Los servidores de aplicaciones y de datos se encargan de asegurar la integridad de los mismos.

Autenticación de usuarios: Solo usuarios con certificados X.509⁵ pueden usar los nodos. Los usuarios deben proveer sus certificados para poder enviar trabajos al planificador.

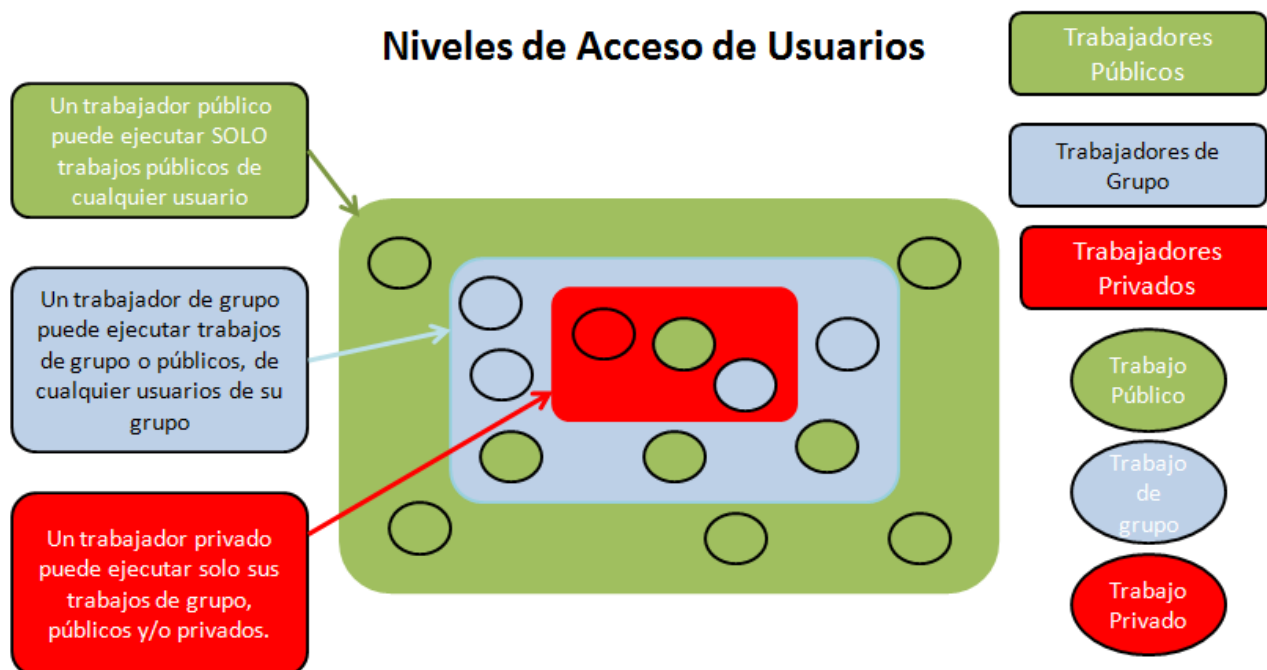
Gestión de usuarios y tareas

Una ventaja significativa que posee este *framework* respecto a los demás es la posibilidad de gestionar permisos para usuarios y tareas como si pertenecieran a un sistema operativo.

Las tareas al igual que los usuarios pueden ser públicos, privados o pertenecer a un grupo.

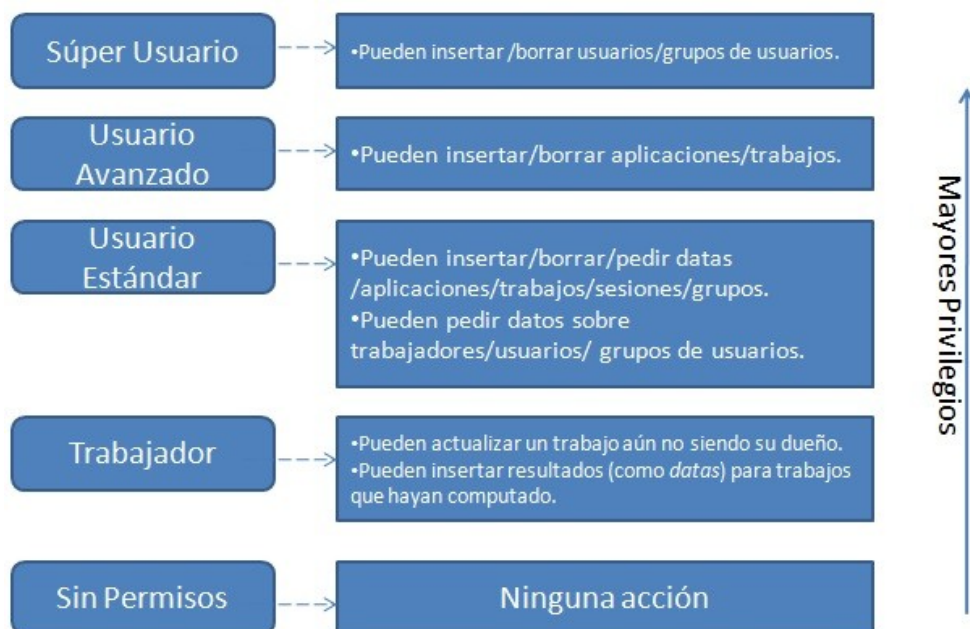
Como se puede ver en la siguiente imagen, los trabajadores públicos solo pueden realizar trabajos públicos. Los trabajadores de grupo solo pueden realizar trabajos de su mismo grupo y públicos. Y Los trabajadores privados pueden realizar trabajos privados, de grupo o público.

5 <http://es.wikipedia.org/wiki/X.509>



Usuarios y sus derechos

Además de los permisos para los usuarios y las tareas, el sistema define una jerarquía de usuarios, definida por la siguiente imagen:



En esta imagen pueden observarse los diferentes tipos de usuarios y sus permisos, nótese que los permisos son escalados es decir los usuarios de arriba heredan todos los permisos de los de abajo.

Cuadros de Sistemas de Cómputo Voluntario

En esta sección veremos dos cuadros comparativos complementarios de los sistemas de cómputo voluntario descriptos hasta el momento. Pensamos que esta información puede ser suficiente y fundamental a fin de facilitar la elección del mejor sistema de cómputo voluntario por parte del proyecto cliente; de acuerdo a sus necesidades específicas.

1er. Cuadro Comparativo de Sistemas de Cómputo Voluntario

	Licencia	Multi Plataforma	Sistema de archivos compartido	Autenticación/ Voluntarios Anónimos	Autorización	X509	Ejecución en Sandbox	Gran Volumen de Datos	Multi Usuarios	Multi Aplicaciones	Experiencia de Voluntarios	Utilizado en Proyectos grandes	Componentes Internos Distribuidos	Comunicación Directa Entre Componentes
BOINC	LGPL	Si	No	Anónimos	No	No	No	Si	No	No, único cliente por PC	Si	Si	No	No
XtreamWeb- CH	GPL	Si	No	Autentica	No	No	No	No	Si	Si	No	No	No	Si
XGrid	Propietaria	No	Si	Autentica	Si	S/D	Si	No	No	S/D	No	No	No	No
GridMP	Comercial	Si	No	Autentica	Si	S/D	Si	No	S/D	S/D	No	No	No	S/D
SLINC	GPL	Si	No	Anónimos	No	No	No	Si	No	No, Único cliente por PC	La que pueda tener con BOINC	No	No	Si, Via RPC
XWHEP	GPL	Si	No	Autentica	Si	Si	Si	Si++	Si	Si	No	No	No	Si

2do. Cuadro Comparativo de Sistemas de Cómputo Voluntario

	Comunicación Directa Entre Componentes	API	Garbage Collector Nativo	Interfaz Amigable	Persistencia	Basado en el sistema de cómputo	Seguridad	Observaciones
BOINC	No	Reducida, CPP y Fortran	DB Purger y File Deleter	Si, Web	Única Bdd: MySQL	Ninguno	Intermedia Mejorable	Arribo de pedidos constante. Muy buena publicidad. Voluntarios en misma PC, no pueden compartir sus archivos. Mala documentación. Tareas poco automatizadas.
XtreamWeb-CH	Si, Via RPC	Amplia: Java, C, CPP, Python	No	Si, Web	Bdd Recomendada: MySQL	XtreamWeb	S/D	Tecnología P2P. Permite la ejecución de aplicaciones paralelas/distribuidas complejas. Descentraliza la arquitectura con respecto a su predecesor.
XGrid	No (Proyecto cliente a controlador a agente)	Reducida	Elimina procesos hijos huérfanos	No, línea de comandos	Vía archivos de proyecto y XMLs	Ninguno	Mem leaks y autenticación no 100% confiable	Preinstalado con MAC Os. Un equipo puede cumplir cualquier rol, sin cambios (servidor/proyecto cliente/voluntario). Poca versatilidad y eficiencia en manejo de datos: Utiliza grandes archivos XMLs.
GridMP	Si	Amplia: Java, C, CPP, Python, Perl, C#	Si, tiene un proceso nativo para eliminar temporales.	Si, Web (MGSI y Consola de Gestión)	DB2 y Oracle	Ninguno	Utiliza firmas digitales y SSL	Restricciones de recursos. Reasignación en caso de falla. Posibilidad de consumir datos en paralelo y empleo de trabajos basados en MPI. Comunicación entre "voluntarios"/"workers".
SLINC	No	Reducida, CPP y Fortran	S/D, elimina los nativos de BOINC	Si, Web	Cualquiera con soporte Hibernate	BOINC	Nula, no fue su objetivo principal.	Instalación y creación de proyectos desde cero, mas rápida que con BOINC. No usa encriptación ni establece ningún tipo de seguridad. No existe la comunicación entre los proyectos cliente y los voluntarios.
XWHEP	Si	Amplia: Java, C, CPP, Python	S/D	Si, Web	Bdd Recomendada: MySQL	XtreamWeb	Comunicación encriptadas. Clave pub/privada	Los recursos son anónimos Es tolerante a Fallas. Monitorea el uso de la PC, y cuando este baja, se activa automáticamente.

Proyectos que estén utilizando esta metodología

Los proyectos más populares de computación distribuida, utilizan la plataforma de BOINC. En esta sección solo describiremos algunos de ellos, demostrando su gran apoyo a la comunidad científica.

GIMPS

El proyecto GIMPS (*Great Internet Mersenne Prime Search*) es considerado como uno de los primeros proyectos de computación voluntaria, y se remonta al año 1.995. Posteriormente se hicieron relativamente famosos otros proyectos como los de *cracking* de algoritmos criptográficos de *distributed.net* en 1.997 o el proyecto *SETI@Home* en 1.999, el cual dió pie al desarrollo de BOINC (*Berkeley Open Infrastructure for Network Computing*).

El proyecto utiliza principalmente el Test de *Lucas-Lehmer*, un algoritmo especializado en el análisis de la primalidad de números de *Mersenne* y especialmente eficiente en arquitecturas informáticas binarias. También dispone de una fase de divisiones sucesivas que tarda horas en vez de semanas y que se emplea para eliminar rápidamente números de *Mersenne* que tienen factores pequeños (que suponen una gran proporción de los candidatos). El proyecto también se vale del algoritmo *p-1* de *Pollard* para buscar factores mayores.

Seti

SETI@Home, permite que cualquier persona con una PC medianamente moderna y una conexión a Internet colabore con la búsqueda de extraterrestres en su tiempo libre.

El proyecto⁶ toma la información obtenida por radiotelescopios y la analiza, buscando patrones que puedan ser identificados como provenientes de una inteligencia extraterrestre.

Es muchísima información, y necesita una supercomputadora para analizarla. En una primera etapa, el proyecto logró contar con 180.000 usuarios activos, que donan una porción del tiempo que el usuario tiene encendida su PC, pero que en la suma de recursos se transforma en una supercomputadora distribuida por todo el planeta. Hoy, la cantidad de usuarios activos sobrepasa el 1.200.000!

WCG:Computing for Clean Water

La misión de *Computing for Clean Water* es proporcionar un conocimiento más profundo sobre la escala molecular en los orígenes del flujo eficiente del agua a través de una nueva clase de materiales filtrantes. Este conocimiento orientará a su vez el futuro desarrollo de filtros de agua económicos y más eficientes.

La falta de acceso al agua limpia es uno de los desafíos humanitarios más grandes para muchas regiones en los países en vías de desarrollo. Se estima que 1,2 mil millones de personas no tienen acceso a agua potable y que 2,6 mil millones carecen de saneamiento. Millones de personas mueren anualmente (se estiman que son 3.900 niños al día) como consecuencia de enfermedades transmitidas por agua no apta para el consumo, especialmente diarrea.

Los científicos esperan usar el conocimiento obtenido de las simulaciones para optimizar el proceso

⁶ <http://setiathome.berkeley.edu>

fundamental que permite que el agua pase mucho más rápidamente por los *nanotubos* y por otros materiales *nanoporosos*. Este proceso de optimización permitirá que el agua pase aún más fácilmente mientras retiene fuentes de contaminación. Las simulaciones también pueden revelar bajo qué condiciones tales filtros pueden ayudar mejor en un proceso de desalinización.

The Clean Energy Project

La misión del Proyecto *Clean Energy* es encontrar nuevos materiales para la próxima generación de células solares y posteriormente, dispositivos de almacenamiento de energía. Al aprovechar el inmenso poder de la *World Community Grid*, los investigadores pueden calcular las propiedades electrónicas de decenas de miles de materiales orgánicos (muchos más de los que jamás podrían ser probados en un laboratorio) y determinar qué candidatos son los más promisorios para desarrollar tecnología de energía solar a bajo costo.

Los investigadores están empleando cálculos de mecánica molecular y de estructura electrónica para predecir las propiedades ópticas y de transporte de las moléculas que podrían convertirse en la próxima generación de materiales para células solares y de combustible.

Ibercivis

Ibercivis es una plataforma de computación ciudadana que permite a la sociedad participar en la investigación científica de forma directa y en tiempo real. El proyecto utiliza BOINC como plataforma de funcionamiento.

Se trata de una iniciativa pionera en España que pretende involucrar al máximo número posible de ciudadanos en la computación voluntaria, que aprovecha la capacidad de cálculo de un ordenador en los momentos en los que está inactivo para realizar tareas derivadas de un proyecto de investigación.

Ibercivis acerca a la ciudadanía investigaciones punteras y la hace partícipe de la generación de conocimiento científico, al tiempo que dota a la comunidad científica de una potente herramienta de cálculo. La PC se convierte en una ventana abierta a la ciencia, creando un canal para el diálogo directo entre investigadores y sociedad.

Dentro de este gran proyecto hay varias investigaciones que se están llevando a cabo de manera simultánea, ellas son *Fusión*, *Docking* (búsqueda de fármacos contra el cáncer), *Materiales* (simulación de sistemas magnéticos), *Neurosim* (una inmersión en la estructura molecular de la memoria), *Nanoluz* (luz a escala nanométrica), *Adsorción* (simulación de fluidos moleculares confinados), *Amiloide* (búsqueda de fármacos contra las enfermedades amiloides neurodegenerativas), *Sanidad* (mejora de diagnósticos), *Ibernet* (donde *Ibercivis* se estudia a si mismo), *Criticalidad* (transporte electrónico en sistemas desordenados con propiedades fractales) y *Cuanticables* (simulación de cables cuánticos).

LHC@home

La mayoría de los problemas de cálculo científico de que los experimentos a los que el *Large Hadron Collider* (*LHC*) se enfrenta requerirán del acceso a enormes cantidades de almacenamiento, el *LHC* producirá 15 Petabytes (15 millones de Gigabytes) de datos por año. Estos requisitos de datos significa que la mayoría de programas no se pueden ejecutar en los ordenadores individuales. Por ello, el CERN está liderando el desarrollo de la computación Grid, que tiene como objetivo

vincular a cientos de centros de computación más importantes del mundo.

Sin embargo, hay excepciones en donde la computación voluntaria tiene sentido para el LHC. En particular, la computación voluntaria es buena para las tareas que necesitan mucha potencia de cálculo, pero la transferencia de datos relativamente poco. En 2004, el Departamento de IT del CERN comenzó a interesarse en probar algunas tecnologías que se utilizan en los proyectos de computación voluntaria como en *SETI@home*. *LHC@home* se convirtió en el título general de estos esfuerzos. Un programa llamado *SixTrack* se ha convertido en la primer aplicación a probarse, y su función es simular las partículas que viajan alrededor del LHC para estudiar la estabilidad de sus órbitas. Esta aplicación fue elegida debido a que puede caber en un solo PC y requiere relativamente poca entrada o salida, pero una gran cantidad de potencia de procesamiento. *Sixtrack* empezó a correr como un proyecto de computación voluntaria en 2004, y ha estado funcionando durante casi todo el tiempo desde entonces. Además, una nueva aplicación llamada *Garfield* ha sido adaptada para que pueda ejecutarse en la misma plataforma BOINC como *Sixtrack*, y se está trabajando en el CERN para ver si software de física más sofisticado puede ser portado, sobre todo utilizando tecnologías de virtualización.

PS3Grid.net

La PS3 es el primer dispositivo que contiene el procesador *Cell* de IBM (en realidad, desarrollado por Sony, Toshiba e IBM). El *Cell* tiene una arquitectura *multicore* especialmente diseñada para optimizar computaciones gráficas. El poder de procesamiento de este chip es enorme, por lo que despertó el interés de varios científicos. Así nació el proyecto *PS3GRID*, con el fin de crear una infraestructura que permitiera unir una colección de consolas PS3 y así lograr un ambiente de simulación molecular distribuido.

La idea es que cualquier usuario pueda descargar y grabar en un *pendrive* USB (de tan solo 1Gbyte), el sistema operativo *Linux Live* y el software *PS3GRID*. Una vez instalado en la *PlayStation 3*, el software se conecta directamente de la consola del usuario al servidor de *PS3GRID* descargando los cálculos científicos que realizará la *PlayStation 3*. Estos cálculos moleculares se realizarán a una velocidad **16** veces superior a la de un PC normal, gracias al procesador *Cell*, sin que el usuario tenga que hacer nada. Para volver a jugar o utilizar la consola con otras finalidades, simplemente debe reiniciar el sistema.

En un primer momento sus autores utilizaron solo la potencia del procesador *Cell*, pero en la actualidad lo han fusionando con la tecnología *GPU* de *NVIDIA* (creando un nuevo proyecto *PS3GRID-GPUGRID*), que supone la infraestructura hecha de varias tarjetas gráficas *NVIDIA* juntas para entregar simulaciones de alta performance. Correr *GPUGRID* en *GPUs* innova en el terreno de la computación voluntaria, al correr aplicaciones que necesitan de alto rendimiento y poder de cómputo en una infraestructura accesible a un precio ultra económico.

EDGeS@Home

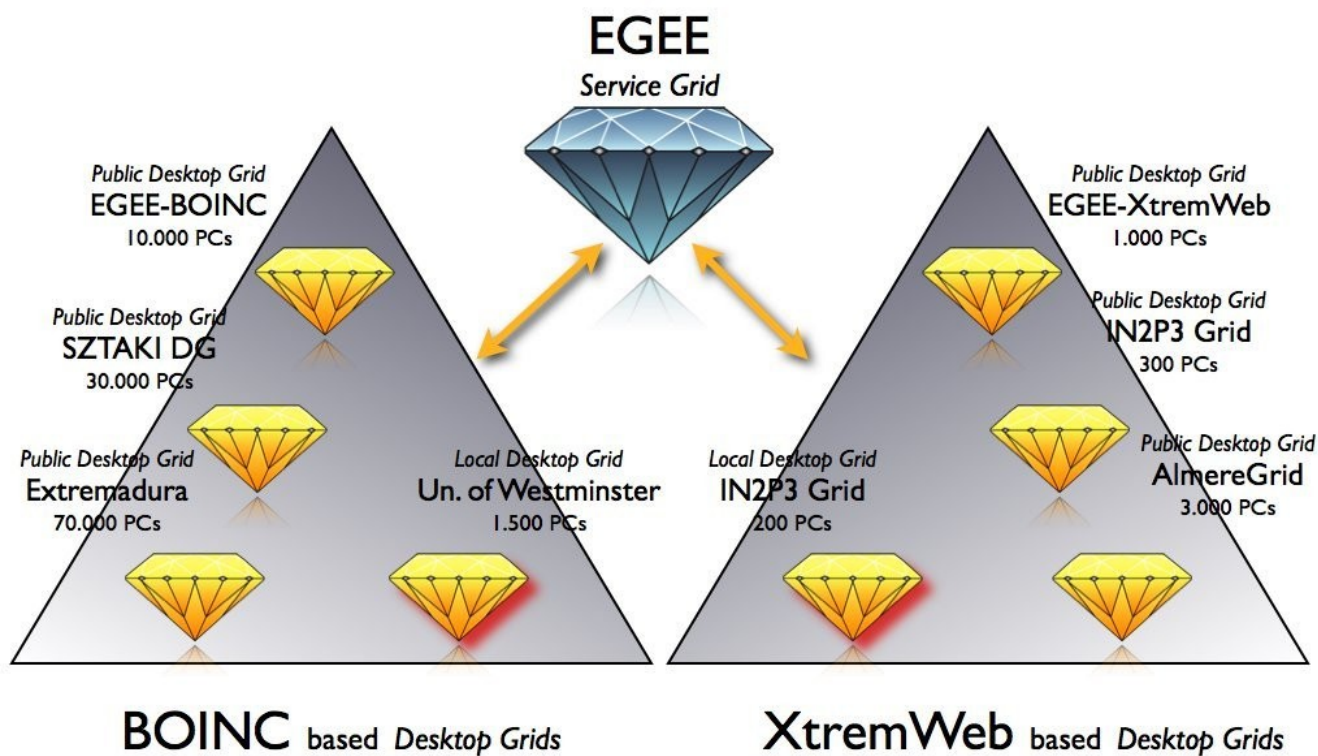
EDGeS@Home es un proyecto multi-plataforma, que lanza aplicaciones desarrolladas por científicos para Grid Computing, pero que también pueden ejecutarse en sistemas de computación voluntaria, como *BOINC* o *XWHEP*.

Cualquier persona puede donar tiempo de computación a *EDGeS@Home*, y por lo tanto a los científicos, usando el servicio *EGEE Grid*. Al igual que los de más proyectos solo es necesario descargar e instalar el cliente en una PC. Cuando el equipo no tiene nada o casi nada que hacer, este cliente pide trabajo a *EDGeS@Home*, ejecuta los trabajos y devuelve los resultados.

EDGE@Home funciona en *Windows*, *MacOSX* y en sistemas *Linux*.

EDGE@Home es seguro: solo se permite el uso de programas probados, validados y fiables para asegurar que ni el software y ni las aplicaciones dañen el equipo de ninguna forma.

Este proyecto posee clientes tanto para XWEP como para BOINC, como podemos observar en la siguiente figura:



El proyecto BOINC en números⁷

En esta sección, se mostrarán estadísticas en cuanto al impacto tiene el proyecto BOINC en número de usuarios, número de recursos, cantidad de países implicados en proyecto, etc. Primero se verá el proyecto en general, y luego será el turno de cada uno de los proyectos de la sección anterior.

Total de recursos (*hosts*) BOINC: 6.232.260

Total de usuarios del *framework*: 2.184.510

Países: 286, Argentina esta en la posición **30** del *ranking* con casi 9.300 usuarios.

Promedio de operaciones punto flotante por segundo: **17,180,706.94 GigaFLOPS (17,180.707 TeraFLOPS)**

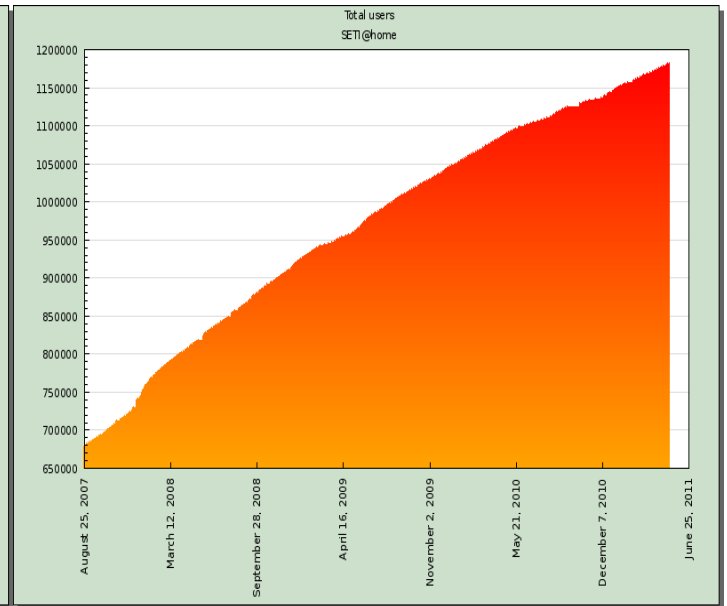
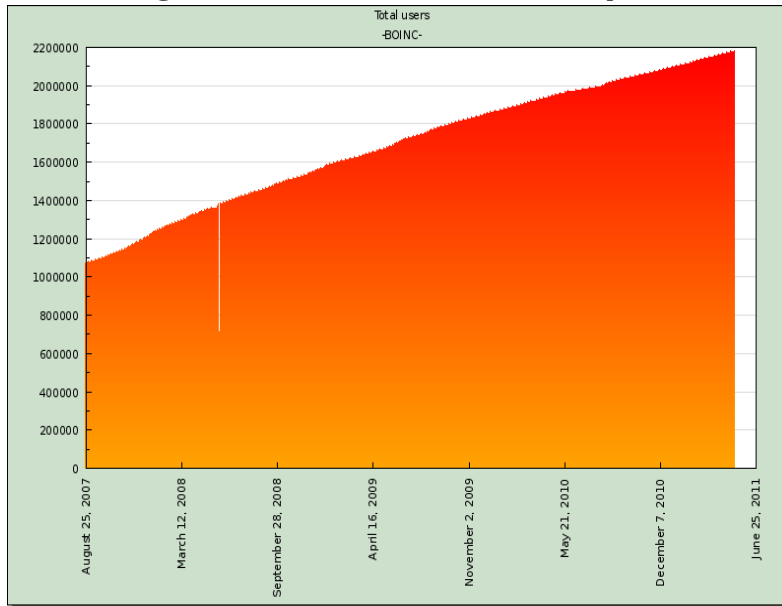
Tabla Comparativa de Estadísticas por Proyecto⁸

Proyecto	Cantidad de Hosts *	Porcentaje de Hosts	Cantidad de Usuarios	Porcentaje de usuarios *	Cantidad de Países	Promedio de oper. punto flotante por seg. (En TeraFLOPS)
BOINC	6232260	100	2184510	100	286	17180.707
GIMPS	439612	7.0538135444	62812	2.87533589	-	59.495
Seti@home	2895496	46.459807518	1183357	54.1703631	253	1571.85
Ibercivis	41609	0.6676390266	16697	0.76433617	134	5
LHC@home	229761	3.6866401594	89032	4.07560506	187	0.093
GPUGrid	22116	0.3548632438	12301	0.56310111	124	606.096
EDGeS@Home	12015	0.1927872072	5069	0.23204288	113	2.991

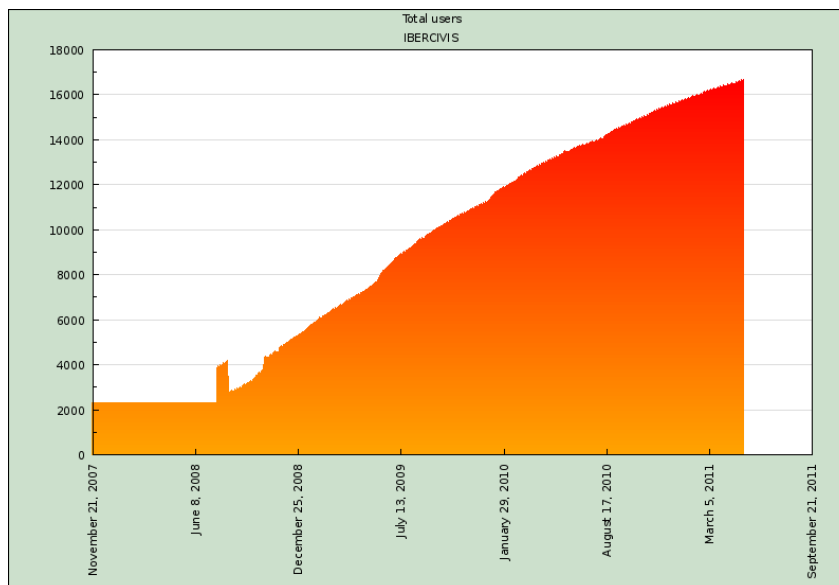
⁷ Actualizado a Mayo de 2011, fuente <http://www.allprojectstats.com>, <http://www.mersenne.org/>

⁸ * El % de hosts y % de usuarios indica cuanta masa voluntaria tiene el proyecto con respecto al total de usuarios de BOINC.

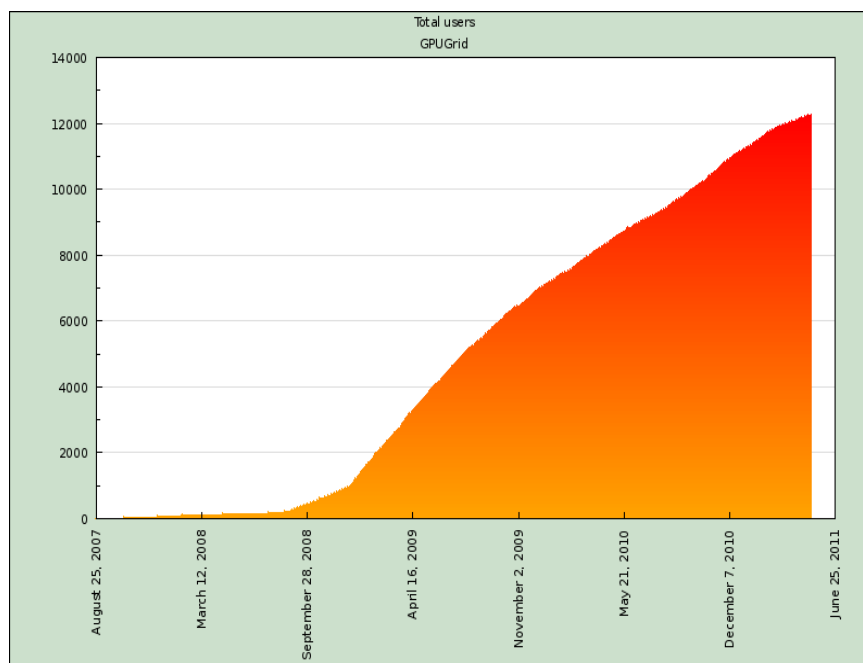
Algunas Estadísticas Representativas



El número de usuarios de Seti, creció en 4 años de 650.000 usuarios, a 1.200.000!



En los inicios, IBERCIVIS era un proyecto meramente Hispano, pero a partir de junio de 2008, la cantidad de usuarios creció a partir de una apertura a nuevos países (de 54 a casi los 140 actuales).



En el gráfico anterior se puede ver que la cantidad de usuarios de PS3GRID (GPUGrid) avanzo a partir de octubre de 2008, gracias a la venta masiva de equipos PS3. Se puede encontrar mas información estadística en <http://www.allprojectstats.com>.

Fortalezas y Debilidades de estos sistemas

En este capítulo, se revisarán las fortalezas y las debilidades de los sistemas de cómputo voluntario. Dada la gran cantidad de proyectos que en la actualidad utilizan *BOINC* y decidimos poner especial énfasis en el análisis de este *framework*. Finalmente, se analizará de manera mas acotada, el *framework Xgrid*. Ambos fueron escogidos, debido a que son de orígenes y licencias distintas. Buscaremos sus puntos fuertes y sus puntos débiles mas importantes, donde podrían mejorar en un futuro.

Fortalezas de BOINC

Buena Publicidad

BOINC es un sistema de cómputo voluntario extremadamente popular. Ésta se basa en la simplicidad de su arquitectura y de su buena publicidad. En los sistemas de cómputo voluntario la publicidad lo es todo, pues se debe conquistar a los usuarios, quienes son los que finalmente deben confiar en la infraestructura subyacente.

Código Abierto

BOINC es distribuido bajo licencia pública *Lesser GNU*. Sin embargo, las aplicaciones que corren sobre la infraestructura *BOINC* no necesariamente deben ser *open source*, dejando abierta la posibilidad a empresas que quieran montar un negocio con proyectos cerrados en torno a su *middleware*.

Random Exponential Backoff

El servidor de tareas de *BOINC* funciona de manera óptima si la tasa de arribo de los pedidos es estable, por lo que si el servidor está caído (por varias horas e incluso días), puede potencialmente saturar a sus clientes enviando muchos pedidos cuando vuelve a entrar en funcionamiento. Para eliminar ese problema, el cliente *BOINC* usa el método llamado *random exponential backoff*; donde el cliente espera por cierto tiempo aleatorio (teniendo en cuenta las veces que fallo el reintento) antes de reenviar el pedido al servidor. Como consecuencia de ello, la tasa de arribo de pedidos se mantiene estable aún habiendo perdido la conexión con el servidor por mucho tiempo.

Soporte para gran volumen de datos

BOINC soporta aplicaciones que producen o consumen un gran volumen de datos, o que usan una gran cantidad de memoria. Los datos pueden estar dispersos en varios servidores, y los voluntarios transferir gran cantidad de datos de manera transparente. Los usuarios pueden especificar límites en cuanto a uso de disco y ancho de banda de la red. Cabe destacar que el trabajo solo es despachado al voluntario si es capaz de manejarlo.

Soporta clientes de múltiples plataformas

El cliente de *BOINC* está disponible para la mayoría de las plataformas:

- *Microsoft Windows* (98 o posterior) corriendo en una CPU Intel x86-compatible.

- Linux corriendo en una CPU Intel x86-compatible.
- Linux corriendo en una CPU AMD x86_64 o Intel EM64T.
- Mac OS X 10.3 o posterior corriendo en Motorola PowerPC .
- Mac OS 10.4 o posterior corriendo en CPU Intel .
- Solaris 2.7 y 2.8 corriendo en una CPU SPARC-compatible.
- Solaris 2.8 o posterior corriendo en una CPU SPARC 64-bits.

Interfaz Web para voluntarios

BOINC provee este tipo de interfaces para que el voluntario cree su cuenta, edite preferencias y cambie su estado. Las preferencias del participante son propagadas automáticamente a todos sus *hosts*. Es decir, que el voluntario solo debe modificar las propiedades de un *host*, para que el cambio se transfiera al resto automáticamente.

Carga de trabajo configurable

El cliente tiene la posibilidad de especificar por cuanto tiempo trabajar o cuantos trabajos realizar, de manera que el voluntario pueda disminuir la frecuencia de conexiones al servidor, o que pueda mantener a su cliente ocupado por un tiempo estable (y no ocioso).

Seguridad de BOINC

El sistema *BOINC* provee mecanismos para reducir la probabilidad de algunos de estos ataques:

Falsificación de resultados y de créditos:

- **Falsificación de resultados.** El atacante retorna resultados incorrectos.
- **Falsificación de créditos.** El atacante retorna resultados argumentando que la computación tomo más tiempo de CPU del que fuera realmente consumido.

BOINC puede detectarlos (y por ende reducirlos) de manera probabilística, utilizando redundancia y verificación de resultados/créditos.

Distribución de ejecutables maliciosos:

- **Ataque:** El atacante entra al servidor de *BOINC*, y modificando la base de datos y los archivos, intenta distribuir su propio ejecutable con virus disfrazado como una aplicación de ciencia.

BOINC usa ejecutables firmados digitalmente para prevenir la distribución de código malicioso.

Aún si los atacantes pudieran entrar en un servidor, si el administrador del mismo tomó las medidas de seguridad sugeridas, el atacante no podrá hacer que los clientes acepten código falso.

Denegación del servicio por ataques al servidor de datos:

- **Ataque:** El atacante repetidamente envía grandes archivos al servidor de datos del proyecto, llenando los discos duros, volviéndolo inusable.

BOINC provee un mecanismo adicional llamado “certificados de subida” para evitar ataques al servidor. Cada archivo de salida posee un tamaño máximo asociado. Además cada proyecto posee un par de claves de autenticación para subir archivos. La clave publica es almacenada en el servidor de datos. Por cada tarea se envía al cliente un descriptor de resultados firmado por el planificador, este descriptor es reenviado por el cliente al servidor de datos cuando el archivo es subido. El

servidor de datos verifica que el descriptor sea verdadero y corresponda con una tarea enviada por el planificador y que el tamaño del archivo sea menor al máximo admitido.

Robo de información de la cuenta de participantes mediante ataques al servidor:

- **Ataque:** El atacante entra al servidor de BOINC y roba direcciones de *email* y otros datos de los participantes.

Cada proyecto debe asegurar la información privada de las cuentas mediante las practicas convencionales de seguridad. Todos los servidores deben ser protegidos por un *firewall* y deben tener todos los servicios de red (que no estén en uso) deshabilitados. Los accesos a esos servidores, debe hacerse solo mediante protocolos encriptados tales como SSH. A su vez, se recomienda que las maquinas sean sujetas regularmente a auditorías de seguridad. BOINC hace especial hincapié en estas practicas, debido a que un ataque exitoso podría desacreditar a TODOS los proyectos que confían su plataforma a BOINC y por ende, la participación publica decaería significativamente.

Abuso intencional de los recursos de los participantes por los proyectos:

- **Ataque:** Un proyecto intencionalmente entrega una aplicación que abusa de los recursos de los participantes, por ejemplo, para robar información sensible guardada en la computadora del participante.

La seguridad del lado del cliente se basa en crear cajas de arena basadas en cuentas de usuarios no privilegiados. Si los permisos de los archivos y los directorios son configurados correctamente, las aplicaciones no tendrán acceso a los archivos fuera del directorio especialmente creado para ejecutar aplicaciones del proyecto. Esto es bueno si el usuario es consciente de ello, dado que el instalador del sistema no presenta **ninguna** advertencia al respecto. (Prueba de ello puede encontrarse en la bibliografía⁹, donde en un extenso foro se habla al respecto y se propone involucrar más al instalador en cuanto a lo que esta configurando).

Abuso accidental de los recursos de los participantes por los proyectos:

- **Ataque:** Un proyecto accidentalmente entrega una aplicación que abusa de los recursos de los participantes, por ejemplo, borrando archivos o causando inestabilidades en la maquina *host*.

BOINC además previene el abuso accidental del uso de los ordenadores de los agentes detectando aplicaciones que utilicen demasiado espacio de disco, memoria o tiempo de cpu y los aborta. Los proyectos pueden minimizar la posibilidad de causar problemas liberando versiones de prueba antes de lanzar las versiones definitivas. Los proyectos deberán probar sus aplicaciones en todas las plataformas y con todos los escenarios de entradas posibles antes de enviarlas a “producción”.

9 http://setiathome.berkeley.edu/forum_thread.php?id=28132

Principales debilidades de *BOINC*

Seguridad de *BOINC*

El sistema *BOINC* no provee ningún mecanismo para mitigar los siguientes ataques:

Robo de información de la cuenta de participantes mediante ataques a la red:

- **Ataque:** El atacante explota vulnerabilidades en los protocolos de red para robar información de los participantes.

BOINC no puede prevenir que los atacantes hagan *sniffing* del tráfico para obtener las claves de cuenta de los participantes y usarlos para obtener mas información del participante.

Robo de archivos de proyectos:

- **Ataque:** El atacante roba archivos de entrada y salida.

Los archivos de entrada y salida usados por las aplicaciones *BOINC* no son encriptados. Las aplicaciones podrían hacer esto por su cuenta, pero hay que tener en cuenta que los datos residen en forma de texto claro en memoria, donde el atacante podría tener acceso fácilmente con una herramienta de *debugging*.

Queremos destacar, que muchas de las fortalezas de *BOINC* son alcanzadas mediante una buena configuración del sistema. Si el sistema esta mal configurado; el participante lamentablemente esta bajo su propio riesgo.

Mala documentación

BOINC es un *framework* para computo distribuido y de código abierto, lo cual ayuda a la comprensión del sistema pero definitivamente esto no es suficiente. En su sitio *web*, existe solo una *wiki* con la documentación del proyecto pero consideramos que la información allí plasmada esta muy lejos de contener lo mínimo indispensable para comenzar un proyecto nuevo de manera ordenada. La información esta demasiado dispersa en múltiples páginas en las que uno termina perdiéndose sin lograr el objetivo. En algunos casos, uno alcanza la información pero volviendo a comenzar la búsqueda desde el principio.

Escalabilidad

Uno de los principales problemas de *BOINC* radica en la base de datos. Esta acumula el estado de todo el sistema y se convierte en el cuello de botella más importante del sistema.

BOINC **NO** balancea la carga ni distribuye sus componentes teniendo en cuenta la localidad espacial ni temporal de datos: Suponiendo que la base de datos puede correr en equipos separados, y que los programas pueden estar replicados, en un *host* con varios procesadores o en diferentes *hosts*; las copias no pueden intercambiar datos entre ellas; es decir, cada copia trabaja con su tarea asignada y no tiene ningún tipo de interacción con el resto.

Tareas poco automatizadas

Cuesta creer que con la gran cantidad de proyectos que utilizan *BOINC*, las tareas de creación de un

proyecto estén tan poco automatizadas. A simple vista parecería ser que la metodología a seguir, es crear un proyecto de demostración y luego adaptarlo a las necesidades del proyecto en cuestión. Nosotros seguimos esos pasos, y pudimos correr, sorteando muchas dificultades, la aplicación de ejemplo. Creemos que el *start-up* de un nuevo proyecto debería ser mas sencillo, teniendo en cuenta que partimos de una maquina virtual con la compilación lista. Nos tomó alrededor de 12 horas (dos personas) setear el entorno, ubicar el código a modificar, compilarlo y correr un proyecto muy similar al de ejemplo.

En el apéndice A se detallan las tareas realizadas para la creación de un proyecto de prueba como demostración de cómo utilizar el *framework*.

API de BOINC

La API que BOINC brinda al público, no es mas que un conjunto de funciones escritas en lenguaje C++ con algunos *wrappers* en FORTRAN, por lo que los componentes nuevos para este *framework* deben, necesariamente, ser escritos en alguno de estos lenguajes. Creemos que es una gran debilidad tener una API acotada sólo a dos lenguajes, y no permite potenciar el hecho de tener el código abierto. Seria interesante brindar una API más amplia, hecho que, sin lugar a dudas, permitiría el ingreso de nuevos desarrolladores de otros lenguajes como Java, Python, etc.

Restricción de usar MySql

BOINC restringe la base de datos a MySQL. Investigando hemos encontrado que la salida de funcionamiento de BOINC (en particular del proyecto seti@home) se produce demasiado a menudo, por fallas en su base de datos. Mantenimientos programados que deberían durar horas, terminan tardando varios días debido a fallas en su base.

El proyecto SETI ha estado fuera de funcionamiento varias veces debido a que la base de datos de BOINC (MySQL) no pudo ser capaz de manejar grandes volúmenes de datos, de manera concurrente. La otra gran fuente de fallas es el hardware, pero la mayoría de las veces es acompañada con corrupción de datos en la misma base de datos.

Colocamos un fragmento de las noticias de SETI de Octubre de 2010:

“The machine that was running the main BOINC database has become too unreliable to use. The backup server does not have the capacity to run the project on its own.”

Basta con mirar un poco las [estadísticas](#)¹⁰ para darse cuenta que tienen un grave problema con la base de datos: se pueden encontrar múltiples *Database server crashed, database is out of space, database outage*, etc.

Hasta tal punto llega la centralización de la base de datos en el proyecto SETI, que cada vez que en Berkeley tienen una falla en el suministro eléctrico, la base de datos queda inaccesible y todos los clientes de SETI en el mundo, no pueden seguir funcionando. Estamos hablando de ventanas emergentes apareciendo simultáneamente en casi 3 millones de *hosts* en 253 países.

Creemos que si bien, BOINC sigue la filosofía *open-source* con su base de datos, la posibilidad de cambiar su base de datos (PostgreSQL¹¹ o la comercial Oracle, por ejemplo, tal como lo hace SLINC) en proyectos *data* intensivos, lo ayudaría a mejorar la experiencia de usuario (en este caso,

10 http://boincstats.com/page/project_news.php?pr=sah_beta

11 <http://www.postgresql.org/>

del voluntario) manteniendo su buena propaganda. Estamos seguros de que si un proyecto hace que el voluntario deba estar chequeando continuamente si el servidor o si la base de datos esta caída, con interacciones seguidas con ventanas emergentes (poniendo de manifiesto la caída de la base de datos), no solo hará que el voluntario no quiera participar más en el proyecto, sino que muy probablemente desinstalará el cliente *BOINC* por completo; perdiendo de esta manera a un voluntario.

Fortalezas de Xgrid

Permisos Unix

Los agentes Xgrid siempre ejecutan sus trabajos como usuario “*nobody*” que, en los equipos configurados correctamente, tiene muy pocos privilegios del sistema y acceso a archivos. Así, el sistema se aprovecha de los permisos Unix para limitar las capacidades de las aplicaciones por usuario. Esto introduce un alto grado de seguridad, siempre y cuando los permisos del sistema de archivos existentes no hayan sido mal configurados (por ejemplo, memorias USB con formato FAT no van a tener permisos de Unix). Por todo esto se recomienda ejecutar “[*repair permissions*](#)”¹² antes de activar Xgrid en cualquier agente.

Ejecución controlada vía sandbox

Todos los trabajos de Xgrid se ejecutan en un entorno limitado o *sandbox*. Este recito reduce aún más los privilegios del trabajo para acceder al sistema. Se pueden asignar permisos para que la tarea solo pueda ver algunas partes del sistema de archivos y eliminar selectivamente el acceso de escritura (independientemente de los permisos de Unix) y limitar otras llamadas a sistema como las comunicaciones entre procesos. Este cordón de seguridad evita además el acceso unidades montadas en Volúmenes, que es donde se suelen montar automáticamente y sin protección memorias USB FAT32. Los ejecutables que tengan activado el bit “*Setuid*” no pueden ejecutarse, haciendo difícil explotar agujeros de seguridad por escalada de privilegios. El agente Xgrid también recoge todos los hijos derivados de un trabajo, por lo tanto no hay manera de que una tarea termine y deje atrás un proceso demonio que continúe ejecutándose.

Debilidades de Xgrid

La cantidad de recursos no puede definirse por adelantado

Las tareas deberían poder determinar cuanta cantidad de recursos necesitan (numero de procesadores, cantidad de memoria, etc). El controlador podría usar esta información para contar las tareas que necesitan múltiples procesadores, de modo de no asignar tareas nuevas a los agentes que las están corriendo. Por otro lado, sabiendo que un agente esta corriendo una tarea CPU intensiva, no asignarle trabajos que requieran una gran cantidad de memoria. En la actualidad, el controlador puede sobrecargar de trabajo un agente y no hay posibilidad de prevenirlo.

Las tareas concurrentes no pueden correr en un mismo agente

Si bien existe una manera de comenzar tareas concurrentemente, no es permitido ubicar esas tareas

¹² http://en.wikipedia.org/wiki/Repair_permissions

en un único agente. Por ejemplo, trabajos altamente paralelizables podrían mejorar su *performance* corriendo en un mismo nodo, sin la sobrecarga de utilizar MPI para la conexión con otros agentes.

La autenticación no es confiable

Los requerimientos para autenticarse por *kerberos* no es aceptable para muchos tipos de agentes, y las contraseñas compartidas en texto plano no proveen un mecanismo demasiado confiable. Xgrid necesita reemplazar las contraseñas compartidas del sistema, por otro sistema más seguro (clave pública/privada *ssh* u otro intercambio público de claves) que permita al controlador tener diferentes claves para cada agente, guardándolas de manera más segura que el formato texto plano.

Problemas si el controlador se desconecta de la red

Xgrid tiene un *bug* asimétrico reconocido. Como es de esperarse, si un agente cambia a estado *off-line*, el controlador puede notarlo y marcarlo como *off-line*. Y cuando el agente vuelve a estar operativo, el controlador lo vuelve a marcar como disponible. Sin embargo si el controlador es desconectado del *jack* de *Ethernet*, aún por solo un segundo, algunos de los agentes pueden fallar en el *handshake* cuando intentan restablecer la conexión. El controlador verá permanentemente estos agentes como *on-line* pero inaccesibles. Si esos agentes son eliminados de la cola de trabajo, no podrán volver automáticamente. Esta condición solo puede corregirse reiniciando el servicio agente (equivalente a un *reboot*) de cada uno de los agentes afectados.

El impacto es muy alto, ya que se necesitan permisos de *admin* para reiniciar los agentes. Si bien *Apple* ha reconocido el *bug*, aún no tiene una solución para el mismo.

Fuga de memoria en el controlador

Sin razón aparente, el controlador requiere un *footprint* de memoria residente comparable a la mitad de la suma de todos los trabajos que puedan estar corriendo. Si se crean muchos trabajos en un mismo directorio, el uso de la memoria se incrementará en el controlador y podrá eventualmente romperlo. Este comportamiento es muy extraño, ya que no hay razón por la que el controlador deba mantener los trabajos en memoria luego de iniciar los agentes. *Apple* no tiene intención de arreglar este *bug*, y sugiere usar un sistema de archivos compartido.

Mejoras a los Sistemas de Cómputo Voluntario

En esta sección, se describirán algunas mejoras que hemos identificado para sistemas de cómputo voluntario en general. Más adelante, propondremos mejoras al sistema BOINC en particular.

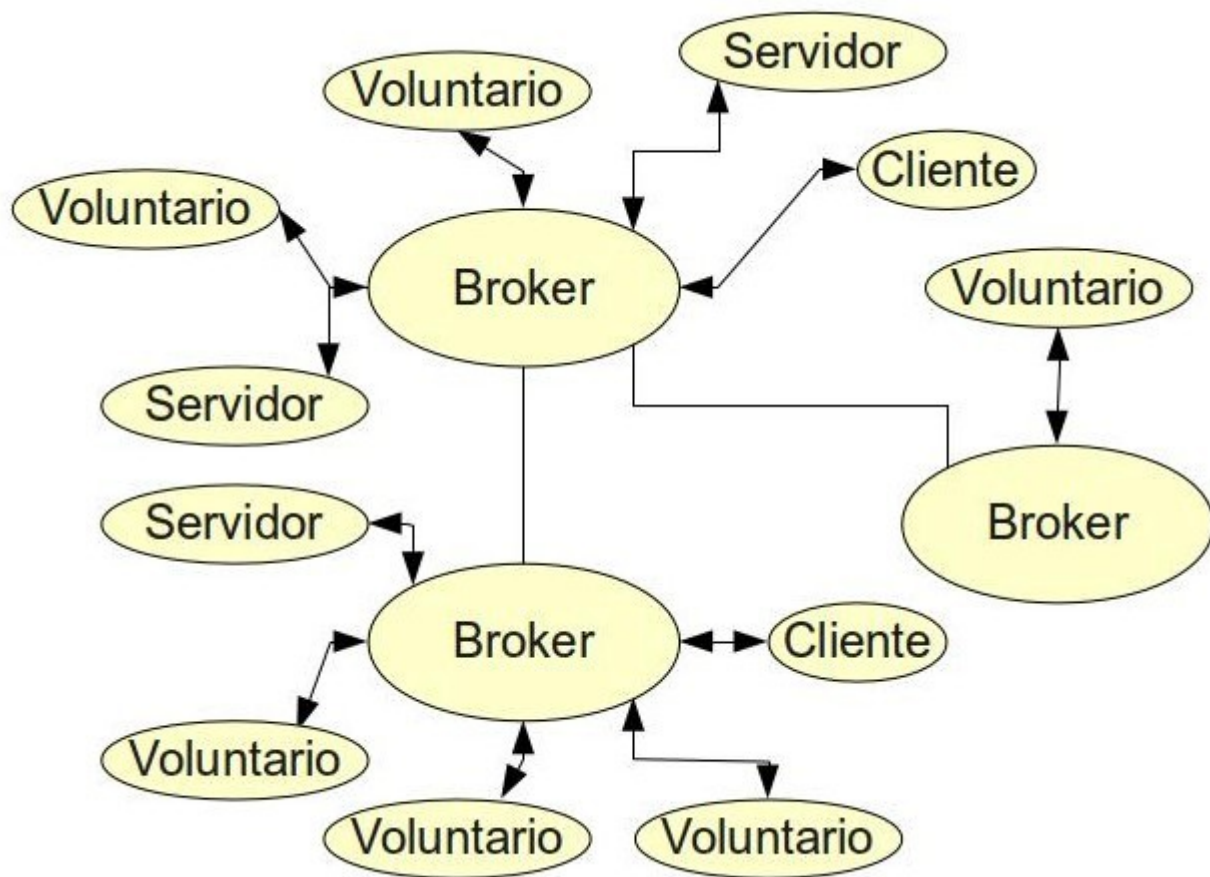
Arquitectura

De los sistemas vigentes, BOINC es el más utilizado. Parecería ser que su simplicidad lo ha hecho muy popular. Sin embargo es esta simplicidad la que lo hace vulnerable. *Boinc* solo puede distribuir parte de sus servicios; debiendo tener una base de datos centralizada. Esta debe ser única, lo que la convierte en el punto débil del sistema. Esto no sólo es un problema por el hecho de que se puede desperdiciar una gran cantidad de cómputo ya conseguido, sino que también desalienta a los voluntarios (tanto del proyecto que tiene problemas con la base, como de otros proyectos que corren sobre la misma plataforma) a seguir usando el sistema. Hemos encontrado diversas entradas en foros donde los usuarios se quejan por las múltiples ventanas emergentes que aparecen cuando la base de datos esta fuera de linea. *XtremWeb for High Energy Physics (XWHEP)* y *XtremWeb-CH* tampoco escapan al hecho de tener un único lugar centralizado para el control de la totalidad del sistema. Estos tipos de sistemas en los que existe un mediador por el que necesariamente tiene que pasar todo el trafico evidentemente no escalaran bien. Por lo que para este tipo de problemas son necesarias arquitecturas de múltiples capas, que aprovechen conexiones p2p entre los interesados y utilicen frameworks que soporten objetos distribuidos. Esto permitirá que nuevos recursos sean agregados de manera dinámica y transparente, lo que a su vez implica una mayor escalabilidad.

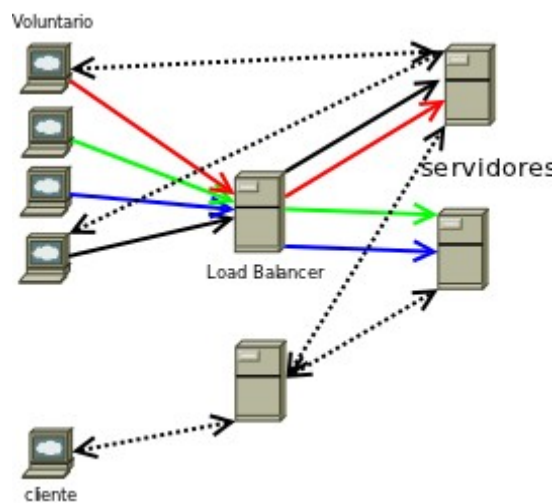
Teniendo en cuenta lo mencionado anteriormente, una mejor alternativa consistiría en utilizar un sistema distribuido en el que cada voluntario no se conecte directamente con el servidor (de datos y de tareas) si no que existan múltiples mediadores (*Broker*) que conozcan las necesidades del voluntario. Cuando los voluntarios se conectan, publican sus intereses (proyectos, requerimientos de sistema, etc). El *broker* les notifica de proyectos en servidores que poseen conjuntos de tareas que cumplen con los requerimientos de los voluntarios. Estos comienzan a hacer *pull* de tareas a los servidores especificados. Si un nuevo proyecto es enviado a un servidor, este notifica al *broker* para que les avise a los voluntarios.

Los clientes, al igual que los voluntarios, se conectan al *broker* para publicar conjuntos de tareas y datos, este ultimo elegirá “el” o “los” servidores que le brinden mejor calidad de servicio y allí publicará tareas y datos. En este modelo, los clientes deben caracterizar las tareas indicando, por ejemplo, los recursos necesarios, el tipo de tareas a realizarse, etc. y los voluntarios deben suscribirse al *broker* indicando los recursos que poseen y las preferencias sobre tipo de tareas que desea realizar (cantidad de memoria requerida por la tarea, tiempo estimado para completar la tarea, etc). De esta manera el servidor que recibió las tareas, las irá publicando y los interesados recibirán la notificación y comenzarán a pedirle tareas.

Al igual que en la arquitectura de XWHEP es conveniente diferenciar los servidores de tareas de los de datos. Existe la posibilidad de que la salida de una tarea sirva como entrada para otra, en ese caso los voluntarios que reciban una tarea que no tenga los datos deberán publicar su necesidad en el *broker*. Cuando un voluntario envíe un resultado, será notificado al *broker* quien, a su vez, notificará a todos los subscriptores para que puedan continuar con la tarea que estaba bloqueada a la espera de nuevos datos. Esta idea es una adaptación del sistema XtremWeb-CH y su proceso espía.



A nivel local los servidores podrían implementarse con un esquema como el que se muestra en la figura en el que existen balanceadores de carga que conocen los nodos disponibles y distribuyen las conexiones entrantes a los nodos ociosos.



Un sistema *map reduce* será el encargado de distribuir las tareas. En la imagen se muestran en líneas punteadas como se distribuyen las tareas desde el cliente a alguno de los balanceadores de carga. Y de este a los servidores de distribución de tareas. El sistema *map reduce* constaría así de dos niveles, un primer nivel que distribuya subtareas a los servidores y un segundo nivel en el que se distribuyen las tareas a los voluntarios.

Una de las alternativas *opensource* que se podría utilizar para implementar esto es Hadoop¹³. Lo que sería necesario realizar, en caso de utilizar esa tecnología, es un nuevo *driver* para un sistema de archivos que tenga en cuenta las características de las redes de voluntarios. Este *file system* debería contemplar una arquitectura de datos similar a la de algunos de los sistemas de cómputo voluntario mencionados anteriormente donde la copia de los datos se hace por duplicado, una copia en un nodo de datos con alta disponibilidad y otra en el mismo nodo donde se generó. Otro tema que debería implementar este *driver* es la posibilidad de brindar comunicaciones punto a punto entre los nodos clientes. Además podría asignarles tamaños de problemas mas grandes a nodos con mayor tiempo de permanencia en el sistema para reducir así la sobrecarga de administración y transferencia de datos.

¹³ Hadoop es un *framework* de *software* libre que soporta aplicaciones *data-intensive* paralelas.

Seguridad

En sistemas de cómputo voluntario, el resultado devuelto por el voluntario es lo que más importancia e interés genera. Es por ello que nos preocupará la seguridad en cuanto a garantizar la integridad de los resultados y la veracidad de los mismos.

Uso de firmas digitales y *checksums*

Los ataques maliciosos o de sabotaje pueden tomar muchas formas. En general, se trata de un nodo malicioso que regresa datos erróneos. Este nodo puede ser un saboteador interno, que es en realidad un voluntario que participó en algún momento del proyecto, o puede ser un Spoofers externo, que es un nodo que no se ha ofrecido voluntariamente, sino que envía mensajes falsos para que parezca que proviene de uno de los voluntarios.

La suplantación de identidad se puede prevenir con firmas digitales. Estas permiten al servidor comprobar que un paquete de resultados proviene fehacientemente de un voluntario. También se puede utilizar en la otra dirección, para asegurar a un voluntario que un *applet* realmente proviene del servidor, y no de un suplantador.

Las firmas digitales pueden ser una solución a ataques externos en sistemas de cómputo voluntario no anónimos, como las redes de voluntarios forzados o “NOIAs”¹⁴. Desafortunadamente, las firmas digitales no son eficaces contra los saboteadores internos, y por lo tanto son inútiles en los sistemas de voluntariado “verdadero” donde se permite a cualquiera, inclusive saboteadores, ofrecerse como voluntarios. Esto se debe a que la firma digital sólo puede autenticar el origen de un paquete de datos, pero nada dice de su contenido.

Una forma de autenticar el contenido de un paquete de datos es incluir un cálculo de suma de comprobación (*checksum*) en el código. De esta manera, si el nodo no ejecuta el código, o se ejecuta incorrectamente, la suma de comprobación no coincidirá, y el servidor puede ser alertado. En la mayoría de los casos, las sumas de comprobación capturan tanto los errores involuntarios, como simples ataques maliciosos tales como devolver los paquetes al azar. También podemos utilizar las sumas de comprobación para autenticar las fuentes (y evitar la suplantación) mediante la transmisión de una clave de control diferente con cada paquete de trabajo. De esta manera, un suplantador externo no conoce la clave correcta a usar, y no puede generar paquetes fraudulentos.

Ambas técnicas solo son útiles contra los ataques maliciosos si los voluntarios están obligados a calcularlas y no se pueden calcular de forma independiente del trabajo. Esto es cierto, por ejemplo, para los NOIAs; donde tanto el *hardware* como *firmware* del nodo puede dificultar que los usuarios desensamblen o modifiquen el código compilado que recibe del servidor. Si bien ayuda a evitar el fraude, no elimina por completo que se pueda desensamblar el código, identificar la firma, generar el código de control, y utilizarlo para “armar” a un paquete de resultados con datos arbitrarios.

Ofuscación

Una forma a garantizar que un nodo no pueda fingir un cálculo de suma de comprobación es evitar que desensamble el código. Si bien no es una tarea sencilla, en algunos casos, puede ser posible mediante criptografía o mediante técnicas de ofuscación dinámica, dificultar la comprensión del código y evitar que pueda aislar el código de control.

14 Network of Information Appliances, un ejemplo de NOIA es el decodificador del televisor.

La ofuscación de código ha sido propuesta por un gran número de investigadores como un medio para dificultar la realización de ingeniería inversa sobre el código. Pero las transformaciones típicas de ofuscación clásica (estática) se basan en razonamientos estáticos sobre ciertas propiedades de los programas, lo cual da como resultado que la ofuscación del código pueda ser fácilmente desentrañada mediante una combinación de análisis estáticos y dinámicos. Por otro lado, la ofuscación dinámica extiende esta idea, ofuscando el código de manera dinámica en el tiempo, agregando el paso de variables entre procedimientos y bloques básicos al de flujo de control del programa, algunos de los cuales nunca van a ser ejecutados. Dichos bloques son tan complejos que no pueden ser determinados *a-priori* mediante análisis estáticos, debido a que las llamadas se generan de manera dinámica durante la ejecución del programa.

Un ejemplo válido de ofuscación dinámica podría ser el variar al azar la fórmula de suma de comprobación y su ubicación en el código compilado del paquete de trabajo. Esto evita que los *hackers* manualmente puedan desensamblar y modificar el código de *bytes*.

El uso de redundancia

Votación por mayoría

El mecanismo más básico de redundancia, es el tradicional método de votación por mayoría. En donde se propone realizar cada trabajo varias veces, y decidir cual aceptar mediante un sistema de voto (el resultado con más cantidad de votos es el que gana). Una forma de aplicar este sistema es mediante una modificación del algoritmo “ansioso” de asignación de tareas (*eager scheduling algorithm*). En la versión normal, el *scheduler* asigna todos los trabajos de una lista circular con una política *round-robin*. Cuando el voluntario completa su trabajo, levanta una bandera para indicarle al *scheduler* que no la asigne a otro voluntario libre.

Si no hay tareas nuevas disponibles, como la lista es circular, el trabajo que fue asignado previamente y no fue completado por un voluntario lento, puede ser reasignado a otros voluntarios ociosos más veloces. Esta asignación de recursos garantiza que voluntarios lentos, no causen cuellos de botella. Si hay un voluntario más rápido que completó su trabajo, puede completar el trabajo de un voluntario mas lento, descartándose la redundancia de resultados.

Cabe destacar, que este tipo de asignación provee una forma básica de tolerancia a fallas, ya que si un voluntario falla o cierra su conexión, dejando su trabajo sin completar, el *scheduler* eventualmente reasigna el trabajo a otro voluntario.

Para el caso de la votación, la bandera que indica que la tarea se completó, no se levanta hasta no encontrar N resultados coincidentes.

Desafortunadamente, la votación también tiene sus inconvenientes. En primer lugar, es ineficiente, tiene un redundancia mínima de 2. Por otro lado, no garantiza de manera absoluta la no falsificación de los datos.

Controles al azar

En los casos donde la tasa de error máximo aceptable no es demasiado pequeño y se busca eficiencia, podemos utilizar una alternativa llamada *spot-checking*¹⁵. En este caso se sabe de ante mano o se puede verificar a posterior el resultado correcto del trabajo que se va a dar a un

¹⁵ Spot-checking = *An inspection or investigation that is carried out at random or limited to a few instances*

voluntario. Entonces, si un trabajador es sorprendido dando un mal resultado, el servidor busca y elimina (vía *backtracking*) todos los resultados recibidos del trabajador. El servidor también puede agregar al usuario a una "lista negra", para que se le invaliden o se le impida la presentación de nuevos resultados.

Respecto de la lista negra, es mejor que los voluntarios que están en ella no lo noten ni se enteren, ya que podrían tomar medidas correctivas. Muchas aproximaciones son analizadas en este sentido, pero lo ideal es seguir enviándole trabajo pero ignorando los resultados. Otra opción es seguir enviándole puntos de verificación para verificar que efectivamente es un nodo fraudulento y no es por una falla momentánea.

Aunque los métodos anteriores son útiles por si solos; la votación y los controles al azar también se pueden combinar para aumentar la fiabilidad. Una forma de hacerlo es aplicar simplemente los dos mecanismos de forma independiente en el mismo sistema.

Para obtener una mayor cobertura y alcanzar tasas de error significativamente mas bajas, se recomienda utilizar una combinación de votación por mayoría, *spot-checking*, "lista negra" y *backtracking*.

Tolerancia a Fallas basada en Credibilidad

Los sistemas de credibilidad computan la probabilidad de que cada resultado tentativo sea correcto basándose en votación (buscando el consenso de los trabajadores) y con controles al azar que le dan más credibilidad a los trabajadores y sus resultados (trabajadores con más credibilidad necesitaran menor consenso). Otros factores como conocimiento de máquinas propias o de confianza que entregan resultados correctos pueden ser tenidos en cuenta en el sistema de credibilidad.

Con este esquema los resultados arriban y se computa la credibilidad del resultado, si ésta credibilidad no supera un cierto umbral, la tarea deberá ser procesada nuevamente. Cuando el nivel de credibilidad es alcanzado, ya sea por que se consiguió el consenso o alguno de los trabajadores paso suficientes chequeos al azar, el resultado es aceptado como resultado final y la tarea se marca como completa.

Esta combinación nos permite manejar la redundancia solo en los casos en que sea necesario, minimizando la cantidad de veces que se debe asignar/procesar una misma tarea.

Mejoras al Sistema BOINC

Muchas de las mejoras a sistemas de cómputo voluntario que hemos identificado con anterioridad, pueden y deben ser aplicadas a BOINC modificando su arquitectura de software inicial para posibilitar:

Involucrar a los usuarios

Uno de los principales problemas de los sistemas de cómputo voluntario es que las personas no saben lo que sucede, ni que hacen, ni para qué. Deben confiar en lo que diga la persona/grupo que publica el proyecto. BOINC no es la excepción. Una posible mejora sería incorporar una plataforma de trabajo colaborativo, donde los voluntarios que estén interesados puedan involucrarse en el proyecto de investigación, no solo aportando su *PC* en momentos ociosos, sino también con conocimiento e ideas nuevas.

Automatizar la creación de proyectos

Como se expuso en secciones anteriores, la tarea de creación de un proyecto desde cero, no es muy intuitivo y para nada automatizado. Solo los proyectos de pensamiento distribuido cuentan con un trato “especial”. BOINC posee un *framework* para "pensamiento" distribuido llamado Bossa¹⁶. Donde se utilizan voluntarios para realizar tareas que requieren de la inteligencia y/o el conocimiento humano. Bossa es un *wizard* que simplifica la tarea de creación de proyectos, de manera simple y poderosa; sin la necesidad de *royalties*, al ser libre y gratuita.

Bossa hasta mantiene un estimado del nivel de *skills* de los voluntarios, asegurando que para cada tarea hay un consenso suficiente en el conjunto de resultados de los voluntarios. El entrenamiento de voluntarios que realicen el trabajo; es casi trivial, utilizando Bolt¹⁷, un *framework* de entrenamiento basado en web que se integra con Bossa.

Creemos que se debería impulsar la creación de este tipo de herramientas (Bossa, Bolt, etc), que permitan tener un proyecto funcionando a partir de una GUI, para minimizar el tiempo total de *start-up* del proyecto.

Extender la API de BOINC

Como se expuso en secciones anteriores, creemos que tener una API acotada sólo a dos lenguajes (C++ y Fortran) es demasiado limitante, por lo que sería interesante brindar una API más amplia, hecho que, sin lugar a dudas, permitiría el ingreso de ideas frescas al proyecto a través de nuevos desarrolladores de otros lenguajes como Java, Perl, Python, etc.

Mejorar la seguridad e Integridad de los datos de BOINC

Creemos que un sistema tan popular como BOINC debería incluir, al menos la posibilidad, de firmar digitalmente tanto las aplicaciones como los resultados; de manera que se permita asegurar al voluntario el origen de la aplicación que finalmente va a correr en su maquina y al servidor comprobar el origen de los resultados.

BOINC solo nos brinda la posibilidad de utilizar redundancia, si bien es mejor que nada; como se

¹⁶ <http://boinc.berkeley.edu/trac/wiki/BossaOverview>

¹⁷ <http://boinc.berkeley.edu/trac/wiki/BoltIntro>

expuso anteriormente, la sobrecarga no es menor y creemos que lo mejor sería implementar alternativas combinadas (*spot-checking*, redundancia, etc.) junto con una autenticación de usuarios. Esto último permitiría caracterizar a los voluntarios, reduciendo la sobrecarga a los “buenos” voluntarios conocidos y eliminando los resultados de los malos voluntarios (“lista negra” y *backtracking*). Ahora bien, si un *hacker* pudiese obtener la clave del usuario, podría no solo robar la información (email, datos personales, etc) sino “ensuciar” (al enviar con datos erróneos) el historial del mismo en el servidor. Para evitar este tipo de ataques, BOINC no provee ningún mecanismo. Creemos que este defecto se podría resolver sin mayores dificultades mediante el uso de conexiones seguras para el envío de claves.

Caracterizar los voluntarios

La política de anonimato de los usuarios BOINC trae acarreados muchos problemas de *performance*; ya que no se puede determinar los recursos con los que cuenta el usuario en ese momento. Nos parece importante que BOINC pueda identificar a los usuarios y a las plataformas donde esta corriendo el sistema. De esta manera, se evitaría enviar aplicaciones demasiado costosas (en cuanto a cómputo o almacenamiento) a usuarios con escasos recursos, y aplicaciones “livianas” a voluntarios con recursos mayores; donde se desperdiciaría *hardware*, posiblemente ocioso.

Para enviar las aplicaciones correctas, el cliente que esta corriendo puede informar de la arquitectura del sistema. Respecto de la *performance* del equipo, la mejor forma de caracterizar a un usuario es enviar una tarea tipo y verificar como se comporta, luego realizar comparaciones relativas a alguna máquina patrón.

Paralelizar trabajos

BOINC supone un único modelo de paralelismo, el paralelismo de datos. BOINC envía exactamente la misma aplicación a cada voluntario (igual para todos) y uno o varios conjuntos de datos (únicos). Teniendo en cuenta la topología del medio de transmisión y la diversidad de los recursos; puede que aplicaciones más complejas necesiten de algún paralelismo tipo “*pipeline*” en el que a los distintos datos se les van aplicando distintas operaciones, obteniendo resultados parciales. El servidor de BOINC luego recabar esa información y puede enviar los datos parciales de las etapas anteriores a los voluntarios que computan la siguiente fase; paralelizando el trabajo pero evitando la conexión directa entre voluntarios.

Retribuir los trabajos

Si bien en la actualidad, los proyectos han implementado listas con los voluntarios que prestan sus recursos por mas tiempo; los saludan en su fecha de cumpleaños, indican cuales son los equipos de trabajo con mas créditos, etc.; creemos que la paga a los voluntarios puede ser un factor decisivo para atraer mayor cantidad de voluntarios al sistema; aún si estos no ganan demasiado dinero, si simplemente pueden seguir usando su ordenador y a un costo casi nulo, no muchas personas desaprovecharían esa oportunidad.

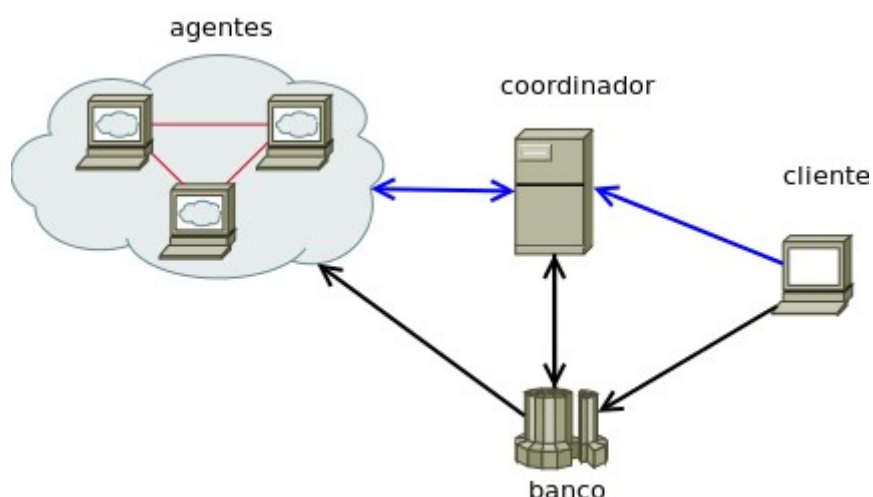
Otros métodos alternativos como sorteos o premios a quienes encuentren los resultados deseados han sido implementados con resultados interesantes, sin embargo no se aplican en todos los proyectos.

Posibilidades de hacer BOINC rentado

Existe la posibilidad de hacer de este tipo de sistemas, un sistema rentado. Aquí se mencionaran dos opciones pero se analizará con mayor detalle solo una de ellas.

La primera de las opciones y la más simple: consiste en que el voluntario no reciba ningún tipo de compensación monetaria. Este caso puede por ejemplo utilizarse para financiar el uso de una aplicación, esto sería, la aplicación es descargada sin costo alguno por el usuario, sin embargo y con el consentimiento del mismo, mientras la aplicación este ejecutándose, también estará corriendo procesos de terceros como lo haría cualquier cliente de un sistema de cómputo voluntario. El creador/vendedor de la aplicación podría cobrar a quienes necesiten del poder de cómputo de sus usuarios y con ello financiar su desarrollo.

La segunda opción que analizaremos con mayor detalle, consiste en que todas las partes implicadas obtengan un beneficio. Tanto los voluntarios como los intermediarios. Para analizar en este caso la posibilidad de hacerlo rentado se propone un esquema como el que se gráfica a continuación:



Donde se observan los siguientes actores:

Agente: un voluntario que ofrece su ordenador a cambio de una retribución

Coordinador: intermediario, que agrupa voluntarios, coordina y distribuye el trabajo enviado por el interesado.

Cliente: una persona física o jurídica interesada en el servicio que brinda el coordinador.

Banco: nombre genérico para denominar a la entidad proveedora de servicios financieros.

El análisis puede ser tan complejo como se quiera o tan simple como asignar montos fijos de dinero a cada tarea.

El cliente deberá proveer de métodos de validación que permitan determinar la veracidad de los resultados. Dependiendo del método de validación y redundancia que se implemente cuando la credibilidad del usuario sea alta y haya entregado suficientes resultados, este podrá cambiar las

tareas realizadas por el dinero ofrecido. Este dinero se le descontará al cliente de su cuenta, el coordinador recibirá una comisión por cada tarea y el voluntario recibirá también su paga.

El usuario deberá evaluar que tareas le convienen más en base a los recursos que ésta necesite y la retribución ofrecida. Así mismo es posible que el sistema analice, mientras ejecuta la tarea, la conveniencia o no de la ejecución de la misma en este sistema y en las condiciones actuales de disponibilidad de recursos del mismo. Esto también obligará al cliente a ofrecer mejores retribuciones, lo que facilitará la adopción de tareas por parte de los usuarios.

Dado el esquema mencionado anteriormente se revisará si es necesario o no hacer algún tipo de modificación a la arquitectura de BOINC.

Seguridad

En este sistema nos encontraremos con todos los ataques ya conocidos a sistemas de *e-commerce*. Sin embargo dejaremos de lado aquellos problemas comunes al *e-commerce* y nos enfocaremos en lo particular de este tipo de sistemas. Para ello analizaremos en que secciones del flujo de tareas en las que podría haber lugar a fallas que posibiliten el uso fraudulento del sistema.

Sobrecarga al enviar trabajos al coordinador

Podría darse el caso de que un cliente envíe tareas al coordinador para sobrecargarlo y no pagar por las tareas que solicita. En este caso se debería implementar un esquema de valoración de los clientes que de manera tal que el planificador de tareas, solo despache tareas de clientes que estén pagando por los resultados que van recibiendo. Se deberá tener en cuenta barajar la posibilidad de falsificación de identidades de los clientes. Por lo que para el envío de tareas los clientes deberían generar peticiones firmadas digitalmente.

Falsificación de la identidad del coordinador

Podría darse el caso de que un coordinador falso este enviando tareas al cliente. Este problema ya esta solucionado por los sistemas de cómputo voluntario utilizando esquemas de claves publica/privada para garantizar la autenticidad de la tarea.

Robo de resultados

Esto se puede resolver utilizando esquemas de clave publica privada para establecer conexiones con el servidor. Y para recibir los resultados del agente a través de conexiones seguras.

Envío de resultados erróneos

El envío de resultados erróneos puede ser evitado de manera suficientemente segura mediante las técnicas descriptas anteriormente en especial la alternativa combinada. Sin embargo se debe tener en cuenta que si la sobrecarga es variable el costo del uso del servicio para los que contraten el uso de la red de voluntarios variará.

Se podrían ofrecer diferentes niveles de servicio, proporcionales a la probabilidad de exactitud de los resultados.

Confidencialidad de la aplicación del cliente

Los clientes pueden desear mantener en secreto las tareas que realiza su aplicación, al igual que los datos de entrada o los resultados. Este sería uno de los problemas más complejos de resolver.

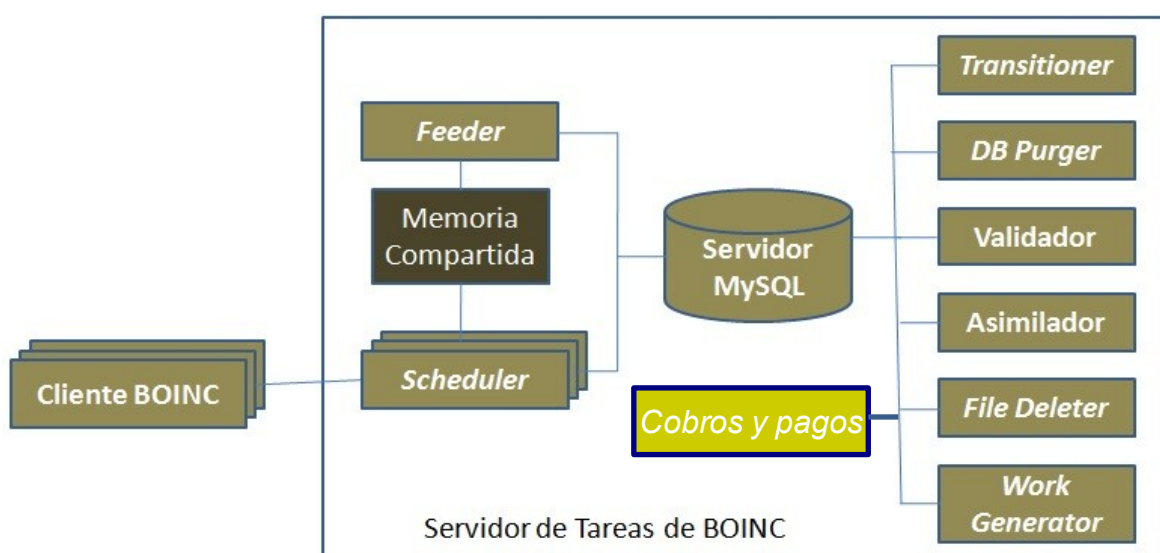
Los servidores tendrán disponible las aplicaciones y podrían copiarlas, desensamblarlas y analizarlas. Esto podría resolverse brindándole al usuario, un servidor al que solo él acceda a cargar su aplicación, y el mediador solo recibiría un *hash* de la aplicación y la firmaría a los fines de garantizar la transparencia. Podría también hacerse lo mismo con los datos. Respecto del lugar donde se realizan los cálculos (las máquinas de los agentes), esto es mucho más difícil pues el agente tendría acceso a todo lo que se ejecute en su máquina.

Una alternativa interesante es la que surge en *paper* de Bayanihan¹⁸ donde se menciona el uso de los NOIA (*network information appliances*) Donde criptografía aplicada por hardware puede ayudar a mantener el control de dispositivo.

Resumen de cambios requeridos por BOINC

La arquitectura de BOINC hace que incorporar nuevas funcionalidades sea relativamente sencillo, y para realizar un proyecto pago no es necesario realizar una modificación sustancial a su arquitectura.

Basta con implementar mejores medidas de seguridad en la conexión. Modificar el módulo de validación para que incorpore mejores algoritmos que aseguren la calidad de los resultados minimizando la redundancia; y que además genere la información necesaria para hacer los cobros y los pagos. La información generada será consumida por un nuevo módulo que evalúe los resultados de cada agente/voluntario y cuando estos alcancen niveles de confianza suficientes, este módulo entregará resultados a los clientes, los cuales a su vez deberán pagar por los mismos. Una vez que el coordinador reciba el dinero, lo deberá distribuir con los agentes que intervinieron en el cómputo.



18 Bayanihan Computing .NET: Grid Computing with XML Web Services
Luis F. G. Sarmenta, et al.

Análisis económico

Si bien este análisis escapa a los alcances de la especialidad se realizará un análisis preliminar, simplificado y con gran cantidad de suposiciones. Se dejará para un trabajo posterior un estudio de mercado más concluyente.

Para que este sistema sea de interés es necesario que tanto los clientes (necesitan resolver un problema) como los voluntarios (poseen los recursos) salgan beneficiados.

Para que les convenga a los agentes, la paga por cada cómputo debería ser mayor al costo de mantener encendido su ordenador. Mientras que para que les convenga a los clientes el costo de enviar su tarea al sistema de cómputo voluntario debería ser inferior al de rentar un sistema de capacidad similar disponible comercialmente.

Dado que es muy complejo hacer una medición del consumo de recursos en el voluntario sin que este pueda adulterarla, se propone usar un sistema en el que a cada tarea se le asigne un valor y luego en el usuario realice una simulación para determinar si le conviene o no ejecutar la tarea teniendo en cuenta sus costos. Matemáticamente esto sería:

$$Paga(tarea) = C_{voluntario} + G_{voluntario} \quad (1)$$

El costo del uso del ordenador del voluntario depende del **tiempo**, el **costo de la pc**, costo del **desgaste** de la PC y el **costo de la energía** eléctrica consumida.

Suponiendo que el costo de la PC no se tiene en cuenta porque el voluntario compra la PC para otro propósito además de ganar dinero con el cómputo voluntario. Suponiendo además que el desgaste de la misma es nulo salvo en aplicaciones que hagan uso del disco rígido. En cuyo caso deberían ser mas valoradas. Suponiendo una vida útil de 5 años¹⁹ (5*8760 h) para un disco con un uso continuo a un costo de \$200²⁰, el costo por hora de uso seria de $\$200/(5*8760 \text{ h}) = 0.00456621005 \text{ \$/h}$

$$C_{voluntario}(t) = CostoDeEnergia[kw/h] * Consumo[w]/1000 + CostoDiscoRigido(t) \quad (2)$$

Ej:

$$\begin{aligned} C_{voluntario}(t) &= 0.5[\text{\$/kw/h}] * 200[w]/1000 * t + 0.0046 * t \\ C_{voluntario}(t) &= (0.1046) * t \end{aligned} \quad (3)$$

donde “t” es el tiempo de duración de la tarea expresado en horas.

El sistema debería calcularle al voluntario una estimación de cuanto le va a costar a el realizar el cómputo solicitado. Luego ese dato debería ser comparado con lo que se le ofrezca de paga y en base a ello solicitarle al usuario que decida si desea aceptar el trabajo o no. Esta decisión en caso de ser revocada debería ser enviada al servidor para que el cliente analice la posibilidad de hacer mejores ofertas. De esta manera el usuario sabrá que mientras utilice el sistema en las condiciones en las que se realizó la evaluación, este le estará reeditando. Además el sistema podría seguir evaluando la conveniencia e indicarle al usuario si en este momento esta o no ganando dinero

¹⁹ De 3 a 5 años es la el tiempo durante el cual las empresas de discos duros ofrecen garantía por los mismos.

²⁰ Precio al 21 de noviembre según mercado libre, para un disco de 160Gb http://articulo.mercadolibre.com.ar/MLA-132633453-disco-rigido-160gb-con-carry-disk-sata-25-_JM

midiendo cuanto se tardó en realizar la tarea efectivamente y cuanto se le pagará.

Si el sistema descubre que el voluntario envió datos erróneos no se le debería pagar ninguna de las tareas que realizó correctamente pues todas ellas deberán ser corroboradas nuevamente. Y debería se bloqueado por fraude.

Será muy importante que los voluntarios no sean bloqueados de manera errónea pues ya no querrán participar. Para esto será necesario incluir un *hash* de verificación para garantizar que los resultados son los que envió el agente.

Respecto del cliente, para analizar la conveniencia, se debe comparar el costo de usar el sistema voluntario contra el costo de rentar suficientes maquinas equivalentes en algún proveedor (como por ejemplo *Amazon*) y crear su propio sistema de cómputo voluntario.

Suponiendo que tiene N tareas a ejecutar que en un ordenador pequeño tardan cada una " t " horas y gestionar esas tareas en el servidor lleva 500 ms^{21} cada una:

$$Paga(tarea) * N + Gservidor(tttotal) + Cservidor(t) > C(rentar en amazon) \quad (4)$$

El costo de utilizar el servidor puede estimarse suponiendo un consumo de 300w constante. Por lo que utilizando la ecuación 2 el costo de utilización del servidor sera de:

$$Cservidor(t) = t * (0.5 * 300 / 1000 + 0.0046) \quad (5)$$

$$Cservidor(t) = t * 0.1546 \quad (6)$$

Si suponemos que solo conseguimos un voluntario con una PC *Windows* y solo alquilamos una PC *Windows* equivalente a la del voluntario en *Amazon*, y suponiendo además que el tiempo total que esta el proyecto en el servidor es " k " veces lo que estaría en *amazon* por la disponibilidad de la PC del voluntario. Suponemos además que tenemos " N " tareas que duran " t " horas. Y tenemos en cuenta además que el costo de rentar en *Amazon* una PC *Windows* pequeña es de $\$0,48$ por cada hora.

$$Paga(tarea) * N + Gservidor + K * t * \$0.1546 < \$0.48 * N * t \quad (7)$$

$$Paga(tarea) * N + Gservidor + K * t * 0.1546 < \$0.48 * N * t \quad (8)$$

Si suponemos una ganancia de $\$0.03$ por tarea del proyecto en el servidor mas $\$0.1$ por hora de estadía.

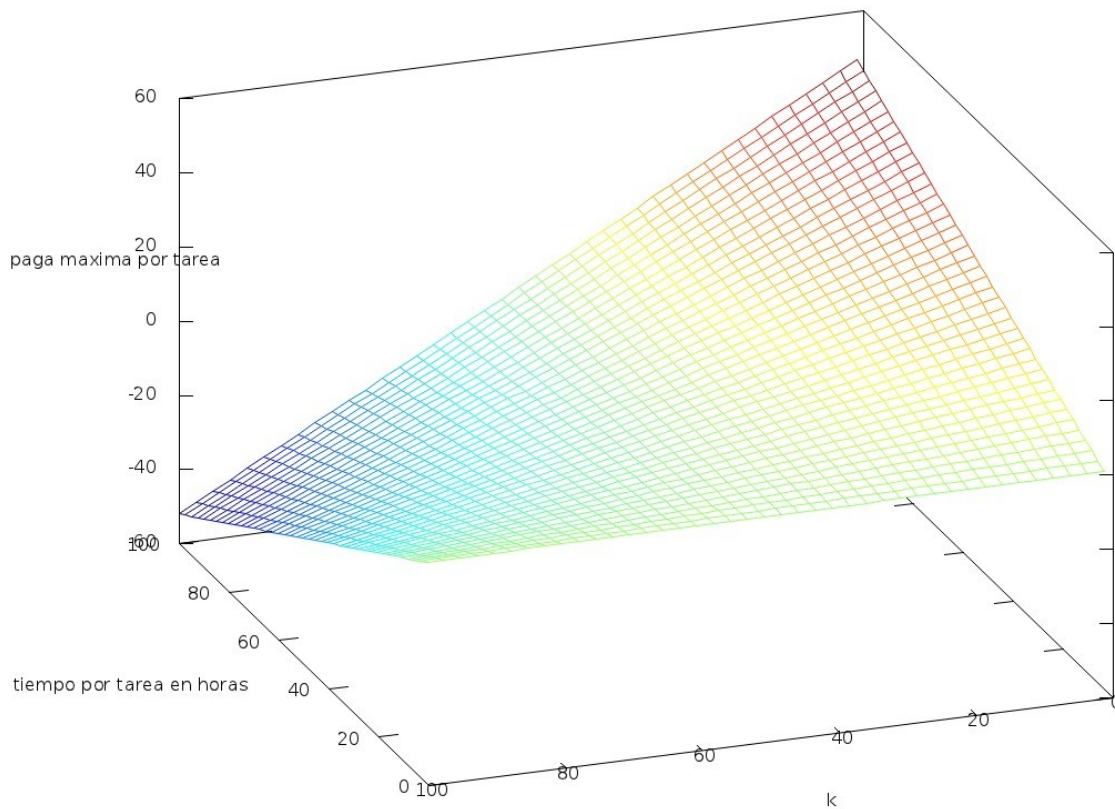
$$N(Paga(tarea) + 0.03) + K * t * (\$0.1 + \$0.1546) < \$0.48 * N * t \quad (9)$$

$$N(Paga(tarea) + 0.03) + K * t * (\$0.2546) < \$0.48 * N * t \quad (10)$$

$$Paga(tarea) < \frac{\$0.48 * N * t - K * t * (\$0.2546)}{N} - 0.03 \quad (11)$$

Como puede observarse en el siguiente gráfico, para la función de costos elegida en la medida que k disminuye y el tiempo por tarea aumenta, se le puede pagar más dinero al agente, las combinaciones que nos interesan son los valores que están en la gama de los verdes y los rojos.

²¹ El tiempo medido durante las simulaciones fue mucho menor, 250ms pero se lo multiplico por un factor de seguridad.



Un tema que podría ser considerado de importancia es la duración de la tarea que contrate el cliente, esta dependerá en gran medida de la cantidad de agentes que deseen realizar la tarea, y de su deseo de continuidad para realizar más tareas. Se podría bonificar el contratar a los agentes por lotes de N tareas o pagar cuando todas las tareas hallan sido finalizadas. En este sentido será de gran utilidad vincular el sistema con las redes sociales para facilitar la publicidad y captación de agentes. Para esto suelen ser de gran utilidad gráficos con los *top ten* de personas que participaron del proyecto o que se beneficiaron con el mismo. *Slogans* como “Gana dinero navegando por internet...” o “Gana dinero chateando ...” serán muy atractivas para todos los que están gran parte del tiempo con su ordenador encendido y reciben una jugosa cuenta de luz cada dos meses.

Otro análisis que interesante es el del la posible ganancia de los agentes, en este caso retomando la cota de la paga, y la definición de costo para el agente nos queda:

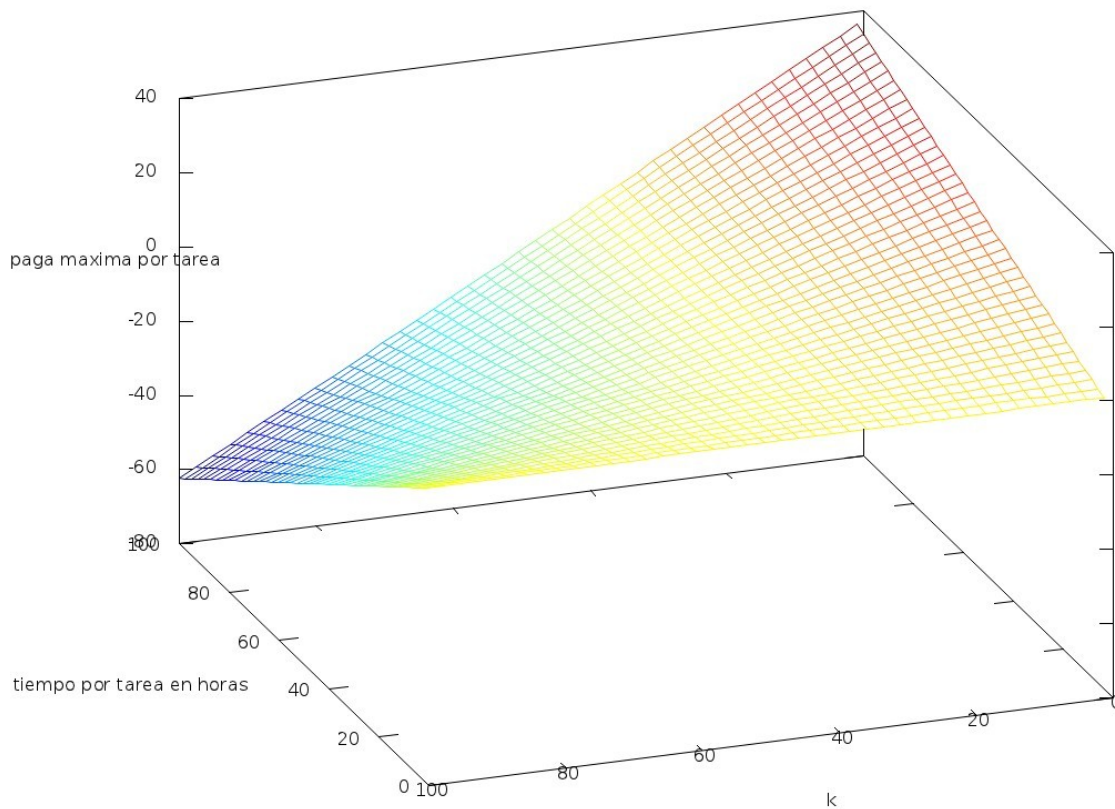
$$\text{Costos} + \text{ganancia} = \text{Paga} \quad (12)$$

$$\text{Paga} < -K * (\$0.01 * t + \$0.00009591) + \$0.48 * t \quad (13)$$

$$(0.1046) * t + g(t) < -K * (\$0.01 * t + \$0.00009591) + \$0.48 * t \quad (14)$$

$$g(t) < -(0.1046) * t - K * (\$0.01 * t + \$0.00009591) + \$0.48 * t \quad (15)$$

$$g(t) < -K * (\$0.01 * t + \$0.00009591) + \$0.3754 * t \quad (16)$$



A continuación se gráfica la máxima ganancia que puede llegar a obtener un usuario. En la imagen toda la zona de la gama de amarillos y rojos.

Para un k razonable de 15 y un tiempo por tarea de 1h.

$$g(t) < -K * (\$0.01 * t + \$0.00009591) + \$0.3754 * t \quad (17)$$

$$g(t) < -15 * (\$0.01 + \$0.00009591) + \$0.3754 \quad (18)$$

$$g(t) < \$0.22396135 \quad (19)$$

por lo que para tareas de 1 hora se podría llegar a ganar hasta 22 centavos y pagar el gasto de energía consumida por el ordenador mientras el voluntario realiza tareas que dejan recursos ociosos.

Conclusiones y Trabajo Futuro

Cada día más y más PC hogareñas cuentan con un increíble poder de cómputo individual, y muchas de ellas están ociosas la mayor parte del tiempo, por lo que consideramos que buscar formas de aprovechar ese poder de cómputo ocioso es siempre algo realmente provechoso y relativamente económico. Luego de realizar la investigación, podemos concluir que el cómputo voluntario simplemente sorprende y asombra día a día. Manejando un volumen de datos y poder de cómputo comparable con grandes *clusters* de súper computadoras, cuesta creer que máquinas hogareñas unidas logren un poder de cómputo cercano a 1.5 petaflops, cuando su dueño esta haciendo un *break*, dejando que funcione el protector de pantallas.

A lo largo del presente trabajo, se han identificado los sistemas de cómputo voluntario más importantes y más populares disponibles; sin hacer ningún tipo de distinción en cuanto a licencias; ya sean *GPL*, *LGPL*, comerciales, cerradas o distribuidas dentro de un SO. Creemos que los objetivos generales de investigación planteados al inicio del trabajo han sido cumplidos, desprendiéndose los siguientes aportes más importantes:

- Se analizaron y estudiaron el estado del arte de los sistemas de cómputo voluntario.
- Se relevaron tecnologías y frameworks utilizados.
- Se identificaron las arquitecturas de cada sistema en particular, describiendo sus componentes principales, su funcionamiento y su ciclo de vida.
- Se realizaron comparaciones de cada uno de ellos mediante un cuadro resumen, con las características más importantes de cada uno.
- Se relevaron los proyectos que estén utilizando esta metodología actualmente.
- Se cuantificaron este tipo de proyectos: poder de cómputo, cantidad de usuarios, países involucrados.
- Se identificaron las fortalezas y debilidades de estos sistemas.
- Se propusieron mejoras tanto a BOINC como a estos sistemas en general.
- Se efectuó un estudio de costos, aunque preliminar, que demostró la factibilidad de hacer rentado este tipo de sistemas.
- En general se estableció el punto de partida, tanto al proyecto cliente que no esta seguro de que sistema de cómputo voluntario seleccionar como a aquel usuario de PC; que quiere compartir sus recursos con algún proyecto de investigación y no sabe como hacerlo.

En cuanto a la elección del mejor sistema de cómputo voluntario, podríamos decir que XWHEP es muy versátil en su arquitectura, muy seguro, y aprovecha los beneficios de P2P; pero su documentación y su propaganda no son muy buenas. Posiblemente tengamos el sistema corriendo en la mejor arquitectura, pero sin una masa critica de voluntarios. En contrapartida, podemos decir que BOINC es el que mejor propaganda tiene, mayor cantidad de voluntarios experimentados y que maneja grandes volúmenes de datos; pero es lento y su base de datos MySQL es el principal punto de fallas (para usuarios de SETI; no sería nada raro); haciendo que el proyecto no progrese y hasta perdiendo voluntarios.

Lo anterior no hace más que ejemplificar la gran heterogeneidad de los sistemas, no puede

identificarse claramente y "*a priori*" el mejor sistema de cómputo voluntario; sino que el investigador (o administrador) encargado del proyecto debe evaluar cada uno de ellos dependiendo la arquitectura que se amolde más a sus necesidades y a los recursos voluntarios con los que disponga.

Trabajos futuros:

Los sistemas de cómputo voluntario han demostrado ser dinámicos y "reinventarse" a partir de sus predecesores; caso XWHEP, XtremWeb-CH y SLINC, por lo que la investigación nunca estará completa. Siempre se esperan nuevos sistemas que abarquen y den solución a nuevas necesidades, por lo que debieran compararse con los anteriores.

Por otro lado, creemos que las mejoras que dejamos aquí planteadas, deberían ser implementadas finalmente en sistemas de cómputo voluntario, para ayudar a la comunidad tanto de investigadores como de voluntarios a tener sistemas más seguros, eficientes y confiables.

Por último, creemos que una investigación profunda del estudio de costos podría beneficiar el trabajo; ayudando a tomar la difícil decisión de pasar de sistemas anónimos y completamente voluntarios; a sistemas rentados por completo o a sistemas híbridos; con voluntarios y "voluntades compradas".

Bibliografía

http://en.wikipedia.org/wiki/Volunteer_computing
<http://www.jgroups.org/taskdistribution.html>
<http://boinc.berkeley.edu>
<http://boinc.berkeley.edu/trac/wiki/SecurityIssues>
<http://boinc.berkeley.edu/trac/wiki/BoincPapers>
<http://www.spiritus-temporis.com/volunteer-computing/>
<http://www.xtremwebch.net/>
<http://tengrid.com/wiki1/index.php?title=Xgrid>
<http://www.univa.com/products/grid-mp.php>
<http://sourceforge.net/projects/slinc/>
<http://ostatic.com/slinc>
<http://dghep.lal.in2p3.fr/lal/doc/xwhep.html>
<http://www.mersenne.org/>
<http://setiathome.berkeley.edu>
<http://cnmm.tsinghua.edu.cn/contents/51/263.html>
<http://cleanenergy.harvard.edu/go/>
http://www.ibercivis.es/index.php?module=public§ion=channels&action=view&id_channel=5&id_subchannel=42
<http://lhcatome.cern.ch/>
http://www.ps3grid.net/pub/ps3grid_chapter.pdf
<http://www.upf.edu/enoticies/es/0708/0705.html>
<http://www.ps3grid.net/>
<http://www.unitedboinc.com/projects/116-ps3grid>
<http://home.edges-grid.eu/home/>
http://setiathome.berkeley.edu/forum_thread.php?id=28132
<http://boinc.berkeley.edu/trac/wiki/SecurityIssues>
http://www.boinc-wiki.info/Security_in_BOINC
http://www.boinc-wiki.info/BOINC_FAQ:_Security
<http://www.allprojectstats.com/>
http://tengrid.com/wiki1/index.php?title=Bugs_and_Missing_Functionality
<http://www.sg.com.mx/content/view/767>

- Advances in Grid and Pervasive Computing: 5th International Conference, Bellavista, Paolo Chang; Ruay-Shiung; Chao, Han-Chieh; Lin, Shin-Feng
- WordWide computing and its applications, Yoshifumi, Takuya Katayama, Michiharu Tsukamoto.
- The Pillars of Computation Theory: State, Encoding, Nondeterminis, Arnold L. Rosenberg.
- Mohamed BenBelgacem, Nabil Abdennadher and Marko Niinimaki: Virtual EZ Grid: A Volunteer Computing Infrastructure for Scientific Medical Applications, International Journal of Handheld Computing Research, 2011 (accepted, forthcoming)
- "RPC-V: Toward Fault-Tolerant RPC for Internet Connected Desktop Grids with Volatile Nodes." - <http://portal.acm.org/citation.cfm?id=1049983>
- Ian Sommerville - Software Engineering 8th Edition.pdf

Apéndice A

Configuración básica de un servidor BINC

La forma mas rápida y recomendada de tener un servidor BOINC funcionando es mediante alguna de las maquinas virtuales Debian especialmente creadas para tal fin que pueden encontrarse en el [sitio del proyecto](#)²².

Para este caso se adjuntará un DVD con una máquina virtual preconfigurada con un programa sencillo creado para este informe a fines de probar la usabilidad y funcionalidad del framework.

Configuración de la maquina virtual:

Ip: 192.168.89.69

Usuarios / contraseñas: boincadm / 12345, root / rootpw

Crear el primer proyecto

Un proyecto BOINC se compone de:

- Una base de datos
- Una estructura de directorios
- Un archivo de configuración (especifica las opciones, demonios y las tareas periódicas a ejecutar).

Varios proyectos de BOINC pueden existir en el mismo *host*. Esto puede ser útil para la creación de proyectos independientes para pruebas y depuración.

La forma más fácil de crear un proyecto nuevo es mediante el script *make_project*, que crea todos los componentes anteriores.

Una vez logeado como *boincadm* ejecutar los siguientes pasos

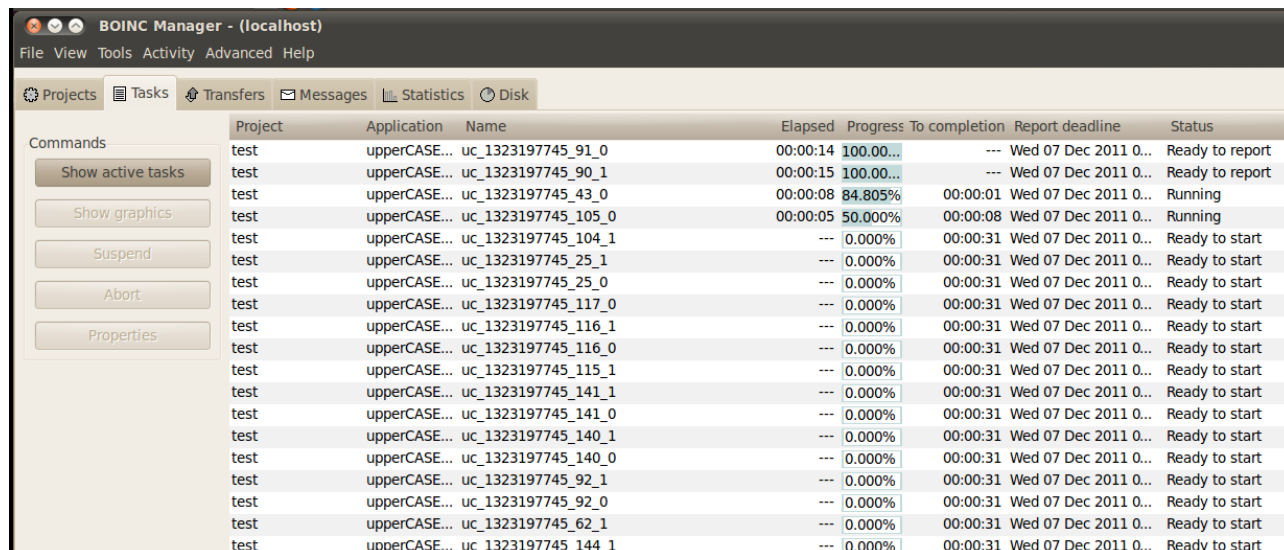
```
$ cd ~/boinc/tools
$ ./make_project --url_base http://192.168.89.69 --test_app test
  (answer "y" to all questions)
$ cd ~/projects/test
$ su -c 'cat test.httpd.conf >> /etc/apache2/httpd.conf'
$ su -c 'apache2ctl -k restart'
$ crontab test.cronjob
$ ./bin/xadd
$ ./bin/update_versions
$ ./bin/start
```

²² <http://boinc.berkeley.edu/trac/wiki/VmServer>

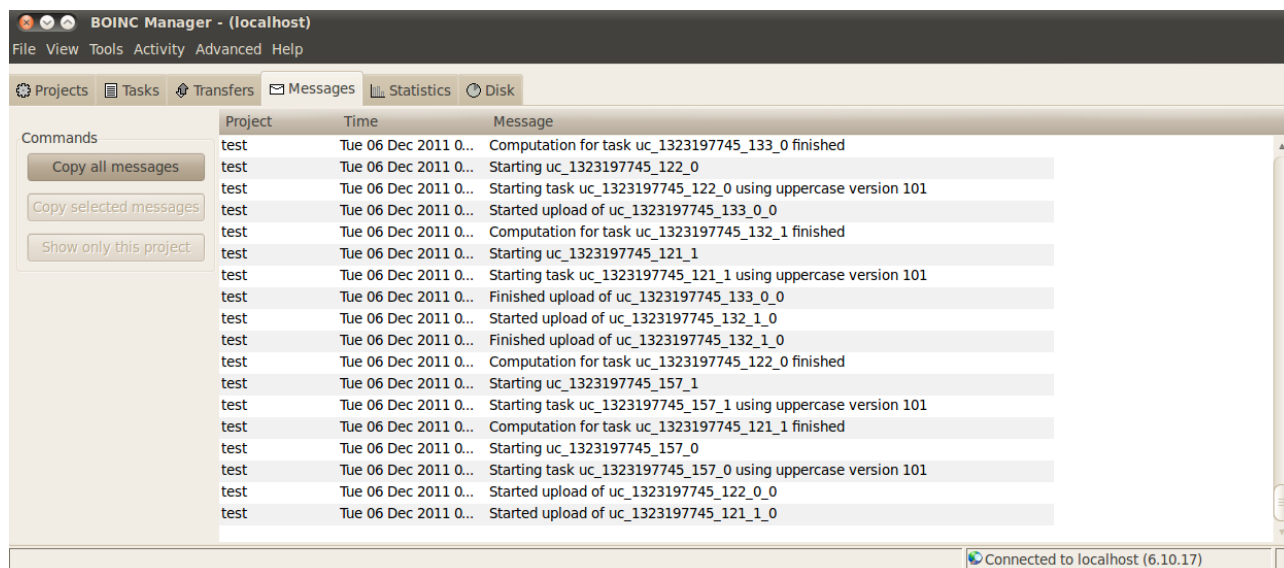
Una vez que realizados los pasos anteriores, descargue, instale y ejecute el cliente BOINC en una computadora Linux²³.

Conecte el cliente para su nuevo proyecto:

Utilizando el Administrador de BOINC, introduzca la URL <http://192.168.89.69/test/>, e ingrese un correo electrónico y contraseña. Una vez que el cliente se pone a trabajar en el proyecto, comenzamos a ver actividad en el *manager*. En la imagen posteriores, pueden verse el estado de las tareas que están corriendo, cuales están por comenzar, cuales finalizaron correctamente y los mensaje:



Project	Application	Name	Elapsed	Progress	To completion	Report deadline	Status
test	upperCASE...	uc_1323197745_91_0	00:00:14	100.00...	---	Wed 07 Dec 2011 0...	Ready to report
test	upperCASE...	uc_1323197745_90_1	00:00:15	100.00...	---	Wed 07 Dec 2011 0...	Ready to report
test	upperCASE...	uc_1323197745_43_0	00:00:08	84.805%	00:00:01	Wed 07 Dec 2011 0...	Running
test	upperCASE...	uc_1323197745_105_0	00:00:05	50.000%	00:00:08	Wed 07 Dec 2011 0...	Running
test	upperCASE...	uc_1323197745_104_1	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_25_1	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_25_0	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_117_0	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_116_1	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_116_0	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_115_1	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_141_1	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_141_0	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_140_1	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_140_0	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_92_1	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_92_0	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_62_1	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start
test	upperCASE...	uc_1323197745_144_1	---	0.000%	00:00:31	Wed 07 Dec 2011 0...	Ready to start



Project	Time	Message
test	Tue 06 Dec 2011 0...	Computation for task uc_1323197745_133_0 finished
test	Tue 06 Dec 2011 0...	Starting uc_1323197745_122_0
test	Tue 06 Dec 2011 0...	Starting task uc_1323197745_122_0 using uppercase version 101
test	Tue 06 Dec 2011 0...	Started upload of uc_1323197745_133_0_0
test	Tue 06 Dec 2011 0...	Computation for task uc_1323197745_132_1 finished
test	Tue 06 Dec 2011 0...	Starting uc_1323197745_121_1
test	Tue 06 Dec 2011 0...	Starting task uc_1323197745_121_1 using uppercase version 101
test	Tue 06 Dec 2011 0...	Finished upload of uc_1323197745_133_0_0
test	Tue 06 Dec 2011 0...	Started upload of uc_1323197745_132_1_0
test	Tue 06 Dec 2011 0...	Finished upload of uc_1323197745_132_1_0
test	Tue 06 Dec 2011 0...	Computation for task uc_1323197745_122_0 finished
test	Tue 06 Dec 2011 0...	Starting uc_1323197745_157_1
test	Tue 06 Dec 2011 0...	Starting task uc_1323197745_157_1 using uppercase version 101
test	Tue 06 Dec 2011 0...	Computation for task uc_1323197745_121_1 finished
test	Tue 06 Dec 2011 0...	Starting uc_1323197745_157_0
test	Tue 06 Dec 2011 0...	Starting task uc_1323197745_157_0 using uppercase version 101
test	Tue 06 Dec 2011 0...	Started upload of uc_1323197745_122_0_0
test	Tue 06 Dec 2011 0...	Started upload of uc_1323197745_121_1_0

²³ El servidor corre y compila sobre una máquina i686 32 bits, al menos en una primera etapa; recomendamos utilizar un cliente en la misma arquitectura.

Personalización de la aplicación a ejecutar en el servidor

La mayoría de los proyectos de BOINC tienen una sola aplicación que hace el trabajo principal del proyecto. A pesar de ello es posible tener varias aplicaciones diferentes para cada proyecto. Por ejemplo, una aplicación podría generar un conjunto de configuraciones iniciales para una simulación, y el otro entonces calcular los resultados de la simulación con las configuraciones previamente generadas.

Básicamente, para crear una nueva aplicación en BOINC son necesarios dos nuevos pasos:

- Creación (adición) de una aplicación en el proyecto BOINC - esto agrega la aplicación a la base de datos del proyecto (pero no la versión particular de la aplicación).
- Actualización de una aplicación existente (lanzando una nueva versión). - Se registra la nueva versión de la base de datos del proyecto, y copia los archivos para la nueva versión desde el directorio/apps . El ejecutable y los archivos adicionales asociados con la versión de la aplicación son firmados y agregados a la zona de descarga.

Sólo se "añade" la aplicación una vez, pero se "actualiza" o "libera" cada vez que se revise el código de su aplicación y desee distribuir una nueva versión a todos los clientes (incluyendo la primera vez que se añada la aplicación). También es necesario "actualizar" la aplicación si desea agregar una versión de una nueva plataforma (Windows, Mac o Linux). A continuación, deberá actualizar las nuevas versiones por separado para cada una de las plataformas.

Agregar una nueva aplicación

Para agregar una nueva aplicación es necesario entrar en el directorio principal del proyecto, luego editar el archivo `project.xml` para agregar un elemento de la forma:

```
<app>
  <name>nombre_de_la_aplicacion</name>
  <user_friendly_name>Nombre amigable de la aplicación</user_friendly_name>
</app>
```

Luego ejecute el comando

```
$ ./bin/xadd
```

el cual introducirá las nuevas entradas del archivo de *project.xml* dentro de la base de datos.

A continuación, cree un directorio nuevo en `project/apps` que es donde las nuevas versiones de la aplicación se tomarán desde el momento en que "liberar" una nueva versión:

```
$ ./project/apps mkdir nombre_de_la_aplicacion
```

Entonces cada vez que desee publicar una nueva versión de su aplicación, incluida la primera, se pone una copia en este directorio y luego se ejecuta el script "*update_versions*", tal como se describe a continuación. El *script* "*update_versions*" copia el archivo de la aplicación desde aquí a la zona de descarga.

Actualización de una aplicación - lanzará una nueva versión

Una vez que se hayan realizado cambios a la aplicación, esta compile sin problemas y se ejecute correctamente de manera independiente (sin BOINC), la misma estar lista para ser lanzada mediante la actualización de la versión de la aplicación.

El nombre completo de la versión de la aplicación tiene tanto el nombre corto de la aplicación como el nombre de la plataforma para la que se construye.

Para liberar una nueva versión del código es necesario poner el ejecutable en un directorio especial para que el *script* de BOINC, *update_versions*, lo pueda encontrar.

Ejemplo:

Copie el archivo ejecutable de las aplicaciones al subdirectorio apropiado usar el nombre de la plataforma completa. Para el "hola, mundo" ejemplo de programa para Linux que diría algo así como

```
$cp hello $PROJECT/apps/hello/hello_6.01_i686-pc-linux-gnu
```

Desde el directorio del proyecto principal de ejecutar el *script* *update_versions* y aceptar los cambios (*commit*).

Durante el funcionamiento normal del feeder, se vuelve a leer la base de datos de una vez por hora (o más frecuentemente cuando se necesita). El *script* “*update_versions*” disparará el “*trigger*” “*reread_db*”, que hace que el feeder automáticamente re-lea la base de datos inmediatamente. Si esto falla por alguna razón, entonces se puede detener y reiniciar el proyecto para forzar el cambio de registro.

Demo: Nueva aplicación a partir de test

Se va a modificar una aplicación en la VM de BOINC, de la manera más simple, tal que el comportamiento del proyecto test (*Uppercase*) cambie por completo su comportamiento.

Hemos decidido que nuestra aplicación sea lo mas simple y didactica posible, por lo que minimizaremos las modificaciones.

Objetivo del proyecto: Dado un par de valores A, B, computar el promedio de ambos.

Vamos a crear un proyecto desde cero, tal como lo hace el “test”:

```
$ cd ~/boinc/tools
$ ./make_project --url_base http://192.168.89.69 --test_app nuestroproy
(answer "y" to all questions)
```

Ahora tenemos un proyecto llamado *nuestroproy*, que tiene el mismo contenido que *test*.

Por un lado, debemos modificar el archivo *uc2.cpp*, de tal manera que acepte dos números y retorne el promedio. Dicho archivo se encuentra en el directorio *boinc/samples/example_app*

Localizar las lineas que van leyendo los caracteres, y van realizando la conversión a *uppercase*²⁴ y

reemplazarlas por el cálculo de promedio:

```
int num1, num2, num_avg;
fscanf(infile, "%d %d", &num1, &num2);
num_avg = (num1+num2)/2;
out.printf("El promedio de (%d,%d) es %d.", num1, num2, num_avg);
```

Los resultados (vía *out*) se manejan mediante una estructura llamada *mfile*²⁵, que permite algunas funciones sobre el archivo de salida (como la *out._putchar(c)*).

Ahora, podemos compilar (*make*, sobre el directorio *boinc/samples/example_app*) y probar el programa de manera *standalone*:

Para ello, debemos crear un archivo de entrada con la estructura: *num1 num2*:

```
$touch in (debe llamarse "in")
$echo -n "10 20" >> in (-n, no queremos el salto de línea final)
$./uc2 in out
$cat out:
El promedio de (10 20) es 15.
```

Este test simple comprueba que el programa, cambió su comportamiento y dado un archivo de entrada tipo rango, computa su promedio.-

Una vez compilado, debe copiarse el archivo binario *uc2* en

projects/test/apps/uppercase/uppercase_1.1_i686-pc-linux-gnu/ uppercase_1.1_i686-pc-linux-gnu

Por otro lado, y paralelamente, se debe modificar el generador de datos, *sample_work_generator* para que ahora genere pares de números. Este archivo se encuentra en el directorio *boinc/sched*.

- Agregaremos tres variables globales enteras: *int intervalomin, intervalomax, delta*;
- Modificaciones en la función *make_job*:
Cambiar: *fprintf(f, "This is the input file for job %s", name);*
por: *fprintf(f, "%d %d", intervalomin, intervalomax);*
intervalomin = intervalomax;
intervalomax = intervalomax + delta;

- Modificaciones en la función *main*:

Antes de llamar a la función *main_loop*, inicializar las variables:

```
intervalomin = 0;
delta = 1000;
intervalomax = intervalomin + delta;
```

Luego correr *make*, sobre el directorio *boinc/sched* y actualizar el proyecto.

- *test/bin/sample_work_generator* debe actualizarse:

```
$cp ~/boinc/sched/sample_work_generator ~/project/nuestroproy/bin/
```

25 http://boinc.berkeley.edu/svn/tags/boinc_core_release_6_11_2/lib/mfile.cpp

sample_work_generator

Continuar con la inicialización del proyecto, tal como lo hacia test:

- `$ cd ~/projects/nuestroproy`
- `$ su -c 'cat test.httpd.conf >> /etc/apache2/httpd.conf'`
- `$ su -c 'apache2ctl -k restart'`
- `$ crontab test.cronjob`
- `$./bin/xadd`
- `$./bin/update_versions`
- `& ./bin/start`

Para compilar la aplicación en Windows es necesario tener instalado Visual Studio 2005 o superior, copiar todo el directorio `/home/boincadm/boinc` en Windows (con nuestro `uc2.cpp` actualizado) y seguir los siguientes pasos:

- Actualizar la carpeta *win_build* (svn update)
- abrir *boinc/samples* con VS y generarla.
- En la carpeta *boinc/win_build/Build/Win32/Debug/* podrá encontrarse el archivo *example_app.exe*
- Para probarlo poner un archivo llamado “in” que contenga dos números separados por un espacio en la misma carpeta. Y comprobar que se cree un archivo llamado “out” con la salida del programa.
- Luego crear una carpeta en el proyecto con el nombre *projects/nuestroproy/apps/uppercase/uppercase_1.1_windows_intelx86.exe/*
- Allí copiar el ejecutable con el nombre “*uppercase_1.1_windows_intelx86.exe*”
- Luego correr `./bin/xadd`
- Por último correr `./bin/update_versions` y si el proyecto ya estaba corriendo solo será necesario poner un nuevo cliente Windows a pedir nuestras tareas.

Con todo esto tenemos una aplicación modificada por nosotros como una prueba de funcionalidad, con ello vimos que la API del sistema no es para nada trivial, y que crear aplicaciones para BOINC no es nada simple, no solo por la complejidad implícita en todo sistema concurrente si no por lo poco ameno de las herramientas que vienen con BOINC. Es necesario realizar demasiado trabajo para crear una aplicación nueva en este *framework*, lo bueno es que junto con el código se proveen una gran cantidad de ejemplos y demonios para llevar a cabo las funciones básicas. Y lo mejor de todo es que al ser este de código abierto, es posible modificarlo y adaptarlo a las necesidades del usuario.



Análisis y Mejoras de Sistemas de Cómputo Voluntario by Javier Jorge, Eduardo Sanchez is licensed under a [Creative Commons Atribución-NoComercial-CompartirDerivadasIgual 3.0 Unported License](http://creativecommons.org/licenses/by-nc-sa/3.0/).

Esta obra está licenciada bajo una Licencia Attribution-NonCommercial-ShareAlike 3.0 Unported de Creative Commons. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-nc-sa/3.0/> o envíenos una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.