UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DESARROLLO DE APLICACIONES PARA SISTEMAS GNU/LINUX EN LA PLATAFORMA DE COMPUTO DISTRIBUIDO BOINC

TESIS

Que para obtener el título de

Ingeniera en Computación

P R E S E N T A

Guadalupe Lizeth Parrales Romay

DIRECTOR DE TESIS

M.C. Alejandro Velázquez Mena

Ciudad Universitaria, Cd. Mx., 2017

AGRADECIMIENTOS

A mi papá y mi mamá porque siempre están conmigo brindándome su amor y su apoyo en todos mis proyectos, sin ustedes cumplir mis metas no sería posible.

A mis compañeros de la Facultad de Ingeniería con los que tuve el gusto de colaborar en el desarrollo de nuestras actividades durante el tiempo que fui alumna en esta facultad.

A mis profesores, por compartir sus conocimientos y sus experiencias.

A mis compañeros de la Administración de los Laboratorios de Computación, en donde realice mi servicio social por su amistad y por compartir sus conocimientos conmigo.

Al M.C. Alejandro Velázquez Mena por darme la oportunidad de participar en el proyecto y por guiarme en el desarrollo de la presente tesis.

Investigación realizada gracias al Programa de Apoyo a Proyectos para la Innovación y Mejoramiento de la Enseñanza (PAPIME) de la UNAM a través del proyecto PE102615, de nombre “Incorporación de las arquitecturas de cómputo distribuido en la asignatura de sistemas distribuidos en la Facultad de Ingeniería”. Agradezco a la DGAPA-UNAM la beca recibida.

Índice

Índice de figuras

Índice de tablas

INTRODUCCIÓN

Desde los inicios de la computación ha existido una demanda creciente de recursos, debido a que los modelos de sistemas dedicados a la investigación consumen muchos recursos tales como tiempo de procesamiento, memoria RAM y espacio de almacenamiento. Además, existen modelos que, programados de manera secuencial son demasiado complejos y de implementarse, comenzarían a arrojar resultados después de años o siglos.

Para atacar estos problemas surgió la programación paralela; a lo largo del tiempo se han desarrollado tecnologías tanto de hardware como de software que han permitido un desarrollo e implementación más eficientes para el modelado de sistemas dentro del campo de la investigación científica y económica, los cuales son sistemas muy robustos y complejos.

El presente trabajo trata sobre el desarrollo de sistemas de cómputo distribuido voluntario, el cual es un modelo de computación paralela en el que se construyen aplicaciones que ocupan el tiempo de procesamiento en desuso de computadoras voluntarias que pueden encontrarse localizadas en cualquier lugar del mundo.

En el primer capítulo se muestra un panorama general sobre las tecnologías de cómputo paralelo más conocidas, se presenta la evolución de los sistemas de hardware paralelo a lo largo de la historia, los cuales se engloban en tres grupos, computadoras paralelas, computadoras vectoriales y clústeres. Se habla también de las tecnologías de software más utilizadas hasta nuestros días. Y por último se muestra un breve panorama de la historia de los proyectos de cómputo distribuido que han surgido desde la década de 1990 hasta nuestros días.

En el segundo capítulo trata sobre las distintas plataformas de computo distribuido que existen actualmente, haciendo énfasis en las características de la plataforma BOINC.

El tercer capítulo presenta el desarrollo del ambiente necesario para trabajar con la plataforma BOINC, comienza por la implementación de un servidor de proyectos, en segunda instancia se habla de las distintas características y componentes de software a desarrollar para implementar para el desarrollo de un proyecto sobre la plataforma BOINC. Se describen los componentes de software y los archivos de configuración a desarrollar para la creación de un proyecto BOINC, y se presenta un ejemplo sobre cómo crear un proyecto de prueba.

En el cuarto capítulo se presenta la configuración de los entornos de desarrollo para arquitecturas Intel, AMD y PowerPC junto con la compilación de una aplicación de prueba para corroborar el correcto funcionamiento de los entornos.

En el capítulo 5 se presentan los casos de prueba Serpent y Euler. El proyecto Serpent se emplea para la simulación de fenómenos físicos que se presentan en reactores nucleares, el proyecto fue portado a BOINC por medio de la aplicación wrapper, de tal forma que no se modificará el código. En el caso de Euler, se trata de un proyecto que se encarga del cálculo de la constante de Euler empleando el polinomio de Newton, para el cual se emplea la librería de precisión arbitraria MPFR.

CAPÍTULO 1: MARCO TEÓRICO

En este capítulo se presenta un panorama general de la computación paralela, tanto como su definición, como su evolución a través de la historia de la computación, se presenta una descripción de los diferentes modelos tanto de hardware como de software. Se presenta la definición de computo distribuido y se define al cómputo voluntario como una forma de computo distribuido, además se presenta una reseña de los primeros sistemas de cómputo distribuido y de algunas de las plataformas de computo distribuido más utilizadas, además se presenta la evolución de BOINC de la Universidad de Berkeley desde sus inicios con el proyecto SETI@HOME.

1.1 COMPUTACIÓN PARALELA

Tradicionalmente el software se programa de forma serial, es decir, se divide el problema en una serie de instrucciones que se ejecutarán una a una de forma secuencial en un solo procesador. [1]

La computación paralela hace uso de múltiples recursos computacionales de forma simultánea. El problema se divide en partes que se pueden resolver de forma concurrente, cada una de las partes se divide en una serie de instrucciones y se ejecuta en un procesador diferente. La principal meta del cómputo paralelo es ejecutar una tarea en el menor tiempo posible, sin embargo, un algoritmo puede no ser paralelizable, y en caso de serlo, también cabe la posibilidad de que paralelizar dicho algoritmo no sea la opción más eficiente. [2]

1.2 HARDWARE

Existen cuatro categorías de computadoras paralelas de acuerdo a la taxonomía de Flynn, que se describen en la tabla 1.1.

Tabla 1.1 Categorías de la taxonomía de Flynn

Categoría	Características
SISD **Single Instruction** **Single Data**	Computadoras con un solo procesador con acceso en un solo programa y almacenamiento de datos. Son computadoras secuenciales convencionales.
MISD **Multiple Instruction** **Single Data**	Computadoras con múltiples procesadores, cada uno de ellos con su propia memoria de programa, con acceso común a una memoria de datos global. Cada procesador toma el mismo elemento de la memoria de datos, pero carga una instrucción posiblemente diferente de su memoria de programa.
SIMD **Single Instruction** **Multiple Data**	Computadoras con múltiples procesadores que tienen acceso privado a una memoria de datos compartida o distribuida y tienen solo una memoria de programa. Cada procesador obtiene la misma instrucción de la memoria de programa, pero obtiene un elemento diferente de la memoria de datos.
MIMD **Multiple Instruction** **Multiple Data**	Computadoras con múltiples procesadores cada uno de los cuales tiene su propio acceso a una memoria de datos y de programa. Cada procesador obtiene instrucciones separadas y datos separados, los procesadores trabajan de forma asíncrona.

En general, existen dos tipos de sistemas MIMD:

- Sistemas de multiprocesadores: aquellos que físicamente utilizan memoria compartida.
- Sistemas de multicomputadoras: aquellos que físicamente utilizan memoria distribuida.

Debido al rápido avance de las tecnologías VLSI, que permiten cada vez más transistores en un circuito integrado, ha sido posible la creación de procesadores multinúcleo, permitiendo el uso de sistemas multicomputadora en los que cada equipo de cómputo es un sistema de multiprocesadores. [2][3]

1.3 HISTORIA DE LA COMPUTACIÓN PARALELA

La computación paralela surgió debido a la necesidad de simular fenómenos que serían peligrosos, extremadamente caros o incluso imposibles de investigar a través de la experimentación. A mediados de la década de los 50's se comienzan proyectos para crear computadoras más eficientes para cubrir las necesidades de computo de proyectos científicos, militares y económicos. Los sistemas con arquitectura paralela tienen sus inicios a partir del diseño de la IBM 704 en 1955, la cual fue la primera computadora comercial con hardware de punto flotante diseñada especialmente para aplicaciones científicas y de ingeniería. En la década de los 60´s y 70´s surgen las supercomputadoras paralelas y las supercomputadoras vectoriales, además surgen ideas que posteriormente se retomaron para el desarrollo de los clústeres en los años 80´s. [4]

1.3.1 SUPERCOMPUTADORAS PARALELAS

Son computadoras en las cuales se emplean varios procesadores que trabajan de forma coordinada, pueden ser computadoras de tipo SIMD o MIMD. [4] En 1956 se crean los proyectos:

1.3.1.1 STRETCH DE IBM

También se conoce como IBM 7030. La finalidad de este proyecto era el desarrollo de una computadora cuyo rendimiento fuera 100 veces mejor que el de las computadoras existentes en esos días. La STRETCH, mostrada en la figura 1.1, se creó bajo un contrato para el Laboratorio Nacional de los Álamos, para la Comisión de Energía Atómica. Empleaba circuitos hechos con transistores, operaba bajo el principio de operación simultánea, es decir, se podía operar en las distintas unidades de almacenamiento con las que contaba al mismo tiempo, lo que permitía un flujo continuo de 2,000,000 instrucciones y 2,000,000 de unidades de datos por segundo. Fue terminada en 1959, en total se construyeron 8 y la

tecnología empleada en su diseño fue utilizada en el desarrollo de la IBM 7090 lanzada al mercado también en 1959. [5]

Figura 1.1 IBM Stretch en operación

1.3.1.2 LARC

Fue una computadora UNIVAC creada por Remington Rand bajo las especificaciones del Laboratorio Nacional Lawrence Livermore (LLNL por sus siglas en inglés); fue entregada en 1960, y la principal función a la que estaba destinada era la simulación de armas nucleares, fue la computadora más rápida de ese año. Era tan grande que se emplearon 5 camiones de 18 ruedas para transportarla hasta el laboratorio, como se muestra en la figura 1.2 y se construyó un edificio para alojarla.

Figura 1.2 Camión que transporta parte de la supercomputadora LARC

Estaba compuesta por cuatro gabinetes; dos de ellos contenían 16KB de memoria de núcleos de ferrita, un gabinete contenía la unidad de procesamiento de entrada/salida, desde donde la información proveniente de las unidades de entrada/salida era enrutada y controlada y una unidad de computo donde la información era procesada. Además, contaba con discos magnéticos, unidades lectoras de tarjetas perforadas y unidades de cinta metálica. [6][7][8]

La LARC implementó tres características de paralelismo nuevas para ese tiempo, en primer lugar, la implementación de dos sistemas de cómputo, como se comentaba anteriormente, uno para el manejo que se encargaba del control y enrutamiento de la información, y otro para el procesamiento de dicha información, en segundo lugar, permitía el manejo simultaneo de instrucciones, registros y operandos, y en tercer lugar, permitía la conexión de otra computadora aritmética, la cual podía ser alimentada por las unidades de entrada/salida. Estas tres características convirtieron a la LARC en la computadora más rápida de su tiempo [9].

Pese a sus características, para el tiempo en que se entregó la computadora al Laboratorio Nacional Lawrence Livermore, sus componentes ya eran obsoletos. [9]

1.3.1.3 ATLAS

Fue una supercomputadora desarrollada en Inglaterra; la Universidad de Manchester con la colaboración de la empresa Ferranti Ltd. desarrollaron el proyecto Atlas (inicialmente llamado MUSE) de 1956 a 1963. La meta del proyecto era crear una supercomputadora que pudiera procesar un millón de instrucciones por segundo, antes de Atlas. Fue considerada como la supercomputadora más poderosa de ese tiempo a nivel mundial. [10]

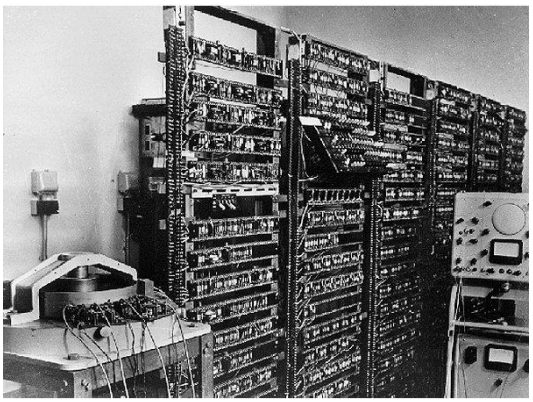


Figura 1.3 Supercomputadora Manchester TC

El equipo de cómputo de la Universidad de Manchester, liderado por Tom Kilburn, construyó una de las primeras computadoras que empleaban transistores de germanio, aunque aún utilizaba tubos de vacío para la lectura y escritura de la memoria y para los ciclos de reloj, la finalidad de este proyecto era crear una versión más pequeña de la Mark I, el resultado fue la Manchester TC (Transistor Computer) que se muestra en la figura 1.3, que comenzó a operar en 1953, posteriormente con la creación de los transistores de juntura se desarrolló una nueva versión de la TC.[11]

Después del éxito obtenido con la TC, el equipo de cómputo de la Universidad de Manchester decidió desarrollar una máquina que pudiera manejar un millón de instrucciones por segundo, inicialmente el proyecto fue llamado MUSE, hasta que la compañía Ferranti Ltd. decidió unirse al proyecto en 1958 y fue renombrado Atlas.

El proyecto concluyó en 1962, y la velocidad de procesamiento alcanzada fue de 630,000 instrucciones por segundo. Una de las características más importantes fue la segmentación de instrucciones que mejoraba no solo el tiempo de ejecución por instrucción, sino los tiempos de ejecución de los programas completos. Atlas además empleaba saltos a subrutinas denominados “extracódigos”, los cuales permitían que instrucciones complejas se ejecutaran como software en lugar de estar incluidas en el hardware.

Figura 1.4 Supercomputadora Atlas

La primer Atlas, la cual se muestra en la figura 1.4, fue puesta en operación el 7 de diciembre de 1962, en la Universidad de Manchester, se estimó que de haber obtenido la supercomputadora con un proveedor externo, hubiera tenido un costo anual de £720,000. Aunque Atlas comenzó a operar en 1962, su sistema operativo llamado “Atlas Supervisor” tardó dos años en ser completamente funcional. [10][12]

1.3.2 SUPERCOMPUTADORAS VECTORIALES

Las computadoras vectoriales pueden trabajar en un vector completo de datos en un ciclo idealmente, a diferencia de las computadoras escalares que trabajan con una sola unidad de datos por ciclo. El rendimiento que proveen se debe en gran medida a su arquitectura con alta segmentación de tal manera que es más eficiente al procesar operaciones que involucran vectores y matrices. [13] [14]

1.3.2.1 PROYECTO SOLOMON

Los primeros trabajos para desarrollar computadoras vectoriales los realizó la empresa Westinghouse, con el proyecto SOLOMON (Simultaneous Operation Linked Ordinal MOdular Network), el cual consistía en desarrollar un sistema que contara con una red de procesadores, denominados elementos de procesamiento (PE) habilitados para comunicarse con los PEs vecinos y con la capacidad de habilitarse o deshabilitarse según los requerimientos del programa que se ejecutara, el diagrama de bloques se muestra en la figura 1.5. Cada PE contaba con su propia memoria, dos registros de trabajo y capacidades aritméticas completas. La red de PEs consistía en 1024 elementos conectados de tal forma que la geometría descrita por la red era un toroide.[15]

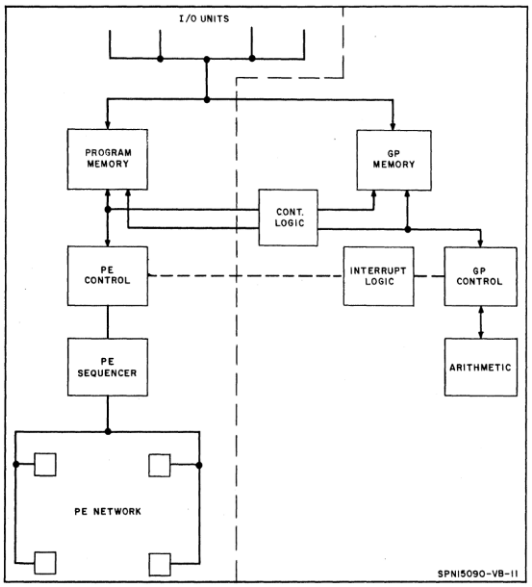


Figura 1.5 Diagrama de bloques de la supercomputadora SOLOMON

El proyecto fue abandonado por Westinghouse y retomado por la Universidad de Illinois que fue financiada por DARPA, el resultado fue la creación de la ILLIAC IV, la Universidad de Illinois consideró viable el proyecto debido a los avances en las

tecnologías LSI y MSI, que en ese tiempo contaban con tiempos de puerta del orden de 2 a 5 ns; además existía la tecnología necesaria para construir memorias con una capacidad de 10^6 palabras con ciclos de 200 a 500 ns a un costo aceptable.

Figura 1.6 Supercomputadora ILLIAC IV

La ILLIAC IV, que se muestra en la figura 1.6, estaba compuesta por una red de 256 elementos de procesamiento acomodados en 4 arreglos de 64 PEs cada uno. Cada PE empleaba operandos de 64 bits y tardaba 240 ns en realizar la operación ADD y 400 ns en realizar la operación MULTIPLY.

Los componentes de la ILLIAC IV fueron desarrollados en diversas partes de Estados Unidos y enviados a la corporación Burroughs en Paoli, Pennsylvania. Fue entregada al Centro de Investigación de la NASA en el sur de San Francisco en 1971. [16][17]

1.3.2.2 CDC STAR-100

Fue un proyecto de la CDC en respuesta a una petición del Laboratorio Lawrence Livermore que requería la construcción de un procesador vectorial capaz de operar a 100 MFLOPS en 1965, la STAR-100 se muestra en la figura 1.7. Se produjeron dos computadoras, la STAR-100A comercializada como CYBER 203 y la STAR100C comercializada como CYBER 205 que llevaron a la construcción de la Cray-1.

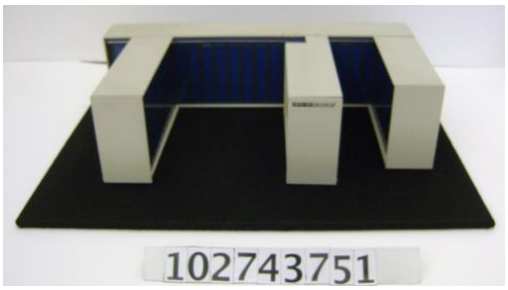


Figura 1.7 Supercomputadora vectorial STAR-100

La principal meta del proyecto STAR-100 era incrementar la velocidad de acceso con el almacenamiento, es decir, memorias RAM de núcleos de ferrita y tecnologías de almacenamiento de materiales semiconductores, entre el procesador y el almacenamiento se implementó comunicación de banda ancha que solo podía aprovecharse empleando pre-fetching de operandos que requería de procesadores que pudieran generar secuencias grandes de direcciones, lo que es característico en procesadores vectoriales. La STAR-100 estaba diseñada para problemas altamente vectorizables, era posible manejar vectores de 65,536 elementos en una sola instrucción, la figura 1.8 muestra a la STAR-100 en operación. [18]


Figura 1.8 Supercomputadora STAR-100

1.3.2.3 CRAY-1

La Cray-1 fue la computadora más rápida entre 1976 y 1982, era 10 veces más rápida que cualquier otra supercomputadora de ese tiempo. La primer Cray-1 fue vendida al Laboratorio Nacional de Los Álamos por $8.8 millones de dólares. Era capaz de procesar 160 millones de operaciones de punto flotante por segundo y poseía una memoria de 8 MB. La construcción de la Cray-1 era en forma de C para permitir que los componentes estuvieran lo más cerca posible mejorando la velocidad del sistema, para combatir la gran cantidad de calor disipada por los

componentes, se diseñó un sistema de refrigeración especial que empleaba gas freón. [19]

Figura 1.9 Seymour Cray junto a la Cray-1

1.3.3 CLÚSTERES

Un clúster es un conjunto de computadoras conectadas a través de una red de datos, como se ilustra en la figura 1.10, y coordinadas a través de un software especial denominado middleware, que se comporta como un único recurso de cómputo. La historia de los clústeres de computadoras comenzó en IBM en la década de 1960 al desarrollar los sistemas HASP (Houston Automatic Spooling Priority) y JES (Job Entry System) que permitían la distribución de trabajo entre clústeres de mainframes, sin embargo, fue hasta la década de 1980 cuando el uso de clústeres cobro verdadera relevancia pues hasta entonces fue posible reunir los elementos necesarios para su desarrollo. Según Yeo y Buyya estos elementos son: los microprocesadores de alto rendimiento, las redes de alta velocidad y el desarrollo de las herramientas estándar para cómputo distribuido de alto desempeño. Además, consideran como un cuarto factor la creciente demanda de poder de procesamiento para la ciencia computacional y para las aplicaciones comerciales [20].

Figura 1.10 Clúster de computadoras

Estos elementos permitieron la construcción de clústeres empleando componentes de bajo costo y fáciles de conseguir tales como las computadoras personales, las workstations y los equipos con multiprocesadores simétricos. Hoy en día los sistemas de cómputo paralelo están migrando de los sistemas especializados, es decir las supercomputadoras, a sistemas de propósito general hechos de componentes débilmente acoplados como los clústeres. El desarrollo de estos sistemas permite la creación de plataformas adaptables a diferentes cargas de trabajo y aplicaciones con un presupuesto relativamente bajo.

El desarrollo de los clústeres fue impulsado por proyectos como Beowulf de la NASA en 1994, Berkeley NOW y HPVM, los cuales demostraron que era posible obtener un sistema con un desempeño igual al de una supercomputadora a un bajo costo, escalable y con componentes independientes de un proveedor. [20]

1.4 SOFTWARE

En la arquitectura de memoria compartida se utilizan APIs como POSIX Threads y OpenMP, mientras que en la arquitectura de memoria distribuida se emplea MPI.

1.4.1 OPENMP

Es un modelo de implementación de computación paralela, se puede implementar con lenguajes como Fortran o C/C++. Está compuesta una librería de subrutinas y por un conjunto de directivas de compilador que describen el paralelismo en el código fuente.

Está diseñado para trabajar con multiprocesadores con memoria compartida. Provee una API estándar y portable que permite el paralelismo incremental. Emplea descomposición funcional. [21]

1.4.2 POSIX THREADS (PTHREADS)

Utiliza hilos para implementar el paralelismo en arquitecturas de memoria compartida en sistemas UNIX a través del lenguaje de programación C, permite un mejor uso de los recursos del sistema, y no utiliza memoria de copia intermedia como lo haría una API de paso de mensajes. Permite el manejo de eventos asíncrono y priorizar tareas. Emplea descomposición funcional. [22]

1.4.3 MPI

Es una librería de paso de mensajes cuya meta es establecer un estándar portable, eficiente y flexible para crear aplicaciones que emplea comunicación por paso de mensajes. Casi todas las plataformas de cómputo de alto desempeño la soportan. No se requiere de grandes modificaciones en el código para migrar entre plataformas. Emplea descomposición de dominio. [23]

1.4.4 JAVA RMI

En el paradigma Orientado a Objetos un programa se encuentra distribuido de manera lógica, lo que facilita el uso de objetos distribuidos entre máquinas virtuales diferentes, invocando los métodos de un objeto de forma remota. Las metas de Java RMI según Menchaca (2001, Art. 3):

- Permitir la invocación remota de objetos que se ejecutan en máquinas virtuales diferentes.
- Dar soporte a las llamadas a los servidores desde applets
- Conservar la semántica de los objetos creados en Java al integrar el modelo de los objetos distribuidos.
- Simplificar el desarrollo de aplicaciones distribuidas.
- Conservar la seguridad que proporciona Java
- Proveer varias semánticas para hacer referencia a objetos remotos. [24]

1.5 SISTEMA DISTRIBUIDO.

Un sistema distribuido está formado por un conjunto de nodos computacionales autónomos coordinados y conectados a través de una red que permite la transferencia de datos con el propósito de realizar una tarea específica común o compartir recursos. Los nodos de un sistema distribuido pueden ser homogéneos (tienen la misma arquitectura y capacidad de cómputo) o heterogéneos (tienen diferentes arquitecturas y capacidades de cómputo), y pueden ejecutar un código

propio para realizar una tarea común o ejecutar una tarea independiente. Los nodos se pueden comunicar por medio de memoria compartida o a través de paso de mensajes. [25] [26]

1.5.1 PRIMEROS PROGRAMAS DE CÓMPUTO DISTRIBUIDO

Los primeros programas de cómputo distribuido fueron el virus Creeper y el software Reaper. Creeper era un programa creado por Bob Thomas y modificado por Ray Tomlinson de BBN Technologies en la década de 1970, estaba diseñado para autoreplicarse recorriendo los nodos de ARPANET empleando el tiempo en desuso de los procesadores, cuando se replicaba en un nodo, se eliminaba junto con todo rastro de su existencia de la computadora anterior, era un programa experimental, el único daño que causaba era impedir la impresión completa de documentos y mostraba un mensaje en la terminal que decía “I´m the creeper: catch me if you can!”. Para eliminarlo se creó el software Reaper, que se encargaba de borrar las copias de Creeper en las computadoras infectadas y también era capaz de replicarse. Se ejecutaba en sistemas Digital PDP-10 que ejecutaban el sistema operativo TENEX. El objetivo de estos programas era experimental, aunque más tarde esta idea serviría para de inspiración para la creación de software malicioso.

Después del desarrollo de Creeper, en 1973, John F. Shoch y Jon A. Hop del Centro de Investigaciones de Xerox en Palo Alto desarrollaron un gusano que se ejecutaba en 100 computadoras conectadas en una red local y empleaba el tiempo en desuso de los CPU’s para procesar gráficos. [27] [28] [29] [30]

1.5.2 DIFERENCIA ENTRE CÓMPUTO PARALELO Y CÓMPUTO DISTRIBUIDO.

El cómputo distribuido requiere de nodos autónomos conectados a través de una red, y que pueden estar distribuidos alrededor del mundo, mientras que el cómputo paralelo no necesariamente requiere de nodos autónomos, como ya se ha explicado anteriormente existen arquitecturas paralelas en las cuales, se tienen varios procesadores dentro de una misma computadora. [31]

1.6 COMPUTO VOLUNTARIO

El computo voluntario también conocido como *peer to peer* o cómputo global es una forma de computo distribuido en la cual, un usuario común puede donar los recursos computacionales de sus dispositivos durante el tiempo en que estos están en desuso a proyectos de computo en su mayoría científicos. La mayoría de estos proyectos tienen restricciones de latencia mínimas, y sus requerimientos de memoria, espacio en disco duro y ancho de banda de red son pequeños. [32]

Los sistemas de cómputo distribuido voluntario surgieron en la década de 1990 cuando el público en general comenzó a tener acceso a Internet.

1.6.1 LA BÚSQUEDA DE NÚMEROS PRIMOS DE MERSENNE

El primer sistema distribuido de cómputo voluntario fue el proyecto de búsqueda de números primos de Mersenne (Great Internet Mersenne Prime Search), su logotipo se muestra en la figura 1.11, el proyecto surgió en enero de 1996, fue creado por George Woltman. El software era una implementación del algoritmo de prueba de números primos Lucas-Lehmer, que se ejecutaba en computadoras con arquitectura Intel i386, y estaba programado en ensamblador.

En ese tiempo no existía una plataforma a través de la cual los equipos unidos al proyecto pudieran solicitar nuevas tareas o enviar los resultados de las tareas procesadas, por lo que se requería que los voluntarios solicitaran las tareas y enviaran los resultados por correo. Los voluntarios elegían un rango de números a probar, y además podían proponer mejoras al código del programa.

Figura 1.11 Logotipo del proyecto GIMPS

Inicialmente el proyecto contaba con poco más de 700 voluntarios; para noviembre de 1996 fue descubierto el 35° número primo de Mersenne y en ese mismo mes, John Sweeney comenzó la implementación del algoritmo en lenguaje C para computadoras PowerMac.

Conforme crecía el número de voluntarios, surgió la necesidad de implementar un sistema más eficiente que la comunicación a través de correo electrónico, por lo que en 1998, con el apoyo de Scott Kurowski de la compañía Entropía, se lanzó PrimeNet, un sistema que se encarga de la comunicación entre los voluntarios, la organización de los nuevos rangos a probar y la publicación de resultados. Hoy en día el proyecto ha logrado encontrar 15 números primos de Mersenne, actualmente existen versiones del algoritmo Lucas-Lehme para GPU's, en la figura 1.12 se muestra el crecimiento de su poder de procesamiento de septiembre de 2016 a mediados de febrero de 2017. [33]

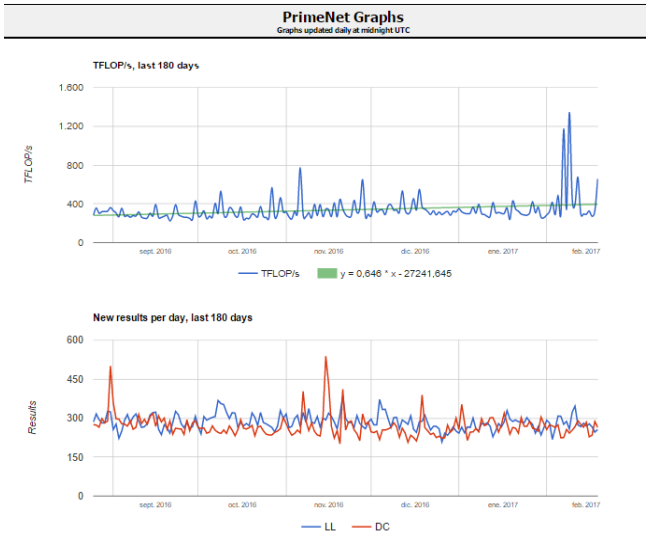


Figura 1.12 Incremento en el poder de procesamiento de GIMPS

1.6.2 LOS DESAFÍOS DE RSA Y DISTRIBUTED.NET

En 1997 los laboratorios RSA, cuyo logotipo se muestra en la figura 1.13, lanzaron el proyecto Secret-Key Challenge, cuya finalidad era cuantificar la seguridad que proveía el estándar de encriptación de datos (DES) con claves de 56 bits y otros cifradores con claves de varios tamaños. En 1999 la maquina "Deep Crack" de Electronic Frontier Foundation en conjunto con distributed.net resolvió el desafío RSA DES Challenge III en 22 horas y 15 minutos.

Figura 1.13 Logotipo de los laboratorios RSA

El desafío Secret-Key de RSA consistió de un desafío DES y doce desafíos del cifrador por bloques de RC5 con claves de tamaño variable. El primer desafío RC5 consistió en un texto plano desconocido encriptado con una clave de 40 bits y el doceavo desafío fue un texto plano encriptado con una clave de 128 bits.

Cada mensaje de texto plano codificado contenía tres bloques de texto que decían “El mensaje desconocido es:” que precedían el mensaje desconocido. La clave de encriptación era elegida al azar y era desconocida para los administradores de los desafíos.

También se crearon trece desafíos de prueba con soluciones conocidas para permitir que los participantes probaran el software. [27] [34]

1.6.2.1 HISTORIA DE DISTRIBUTED.NET

En un inicio, en febrero de 1997, Early Ady de los laboratorios New Media creó el proyecto genx, que consistía en un sistema servidor de claves y software cliente dirigido a algunas plataformas, el código del servidor fue desarrollado por Christopher Stach y probado y optimizado por Roman Gollent, era posible procesar máximo 500 Mclaves/segundo.

El servidor genx.net sufrió ataques de inundación SYN, por lo que Jeff “Bovine” Lawson desarrollo un sistema de buffer proxy para que los voluntarios pudieran seguir ejecutando el cliente aún si el servidor no estaba disponible. En conjunto con genx.net, el proyecto de Bovine (distributed.net) se logró procesar a una tasa de 28.36 Mclaves/segundo, con los clientes conectados en ese entonces, se tenía contemplado un tiempo de 80.6 años para completar el desafío RC5-56. En ese momento no se llevaba un registro del trabajo completado, solo se almacenaba en el servidor, y las claves eran asignadas a los clientes en forma aleatoria.

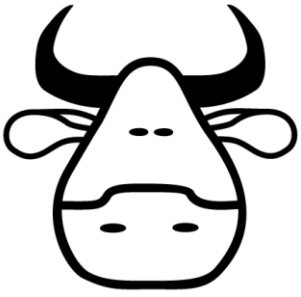


Figura 1.14 logotipo de Distributed.net

Para marzo de 1997, genx.net dejó de operar y la red proxy de distributed.net, cuyo primer logotipo se muestra en la figura 1.14, se hizo pública, además el envío de las llaves comenzó a hacerse en forma ordenada, y esta vez sí se realizaba un registro de las claves utilizadas. Para el 17 de abril distributed.net operaba a una tasa de 100 Mclaves/segundo, lo que redujo el tiempo de espera para completar el desafío a 22.1 años. Se implementaron bases de datos de estadísticas en varios de los servidores de claves, aunque no existía un reporte de las estadísticas globales.

El 8 de mayo se registró el dominio distributed.net en InterNIC, y comienza el proyecto DuncanStats que mostraba las estadísticas globales del proyecto. El 8 de julio se lanzó la versión 2 del cliente, que contaba con un menú de configuración, permitía que se suspendiera el trabajo con señales de suspensión, como ctrl-c, HUP, etc. Además, permitía conocer el porcentaje de trabajo realizado, buscaba una nueva conexión con un proxy si alguno caía, se comenzó a reportar la arquitectura del procesador de las máquinas cliente y el sistema operativo que ejecutaban, hubo una mejora en la velocidad de procesamiento, por ejemplo, en arquitecturas x86 la velocidad aumento entre 20 y 50%.

Para el 2 de septiembre se había alcanzado una tasa de 3.3 Gclaves/segundo y se estimaba que se terminaría de explorar el espacio de llaves del desafío RC5-56 en 201 días, para entonces se había explorado el 20.51% de dicho espacio. El 22 de octubre se completó el desafío RC5-56 después de 212 días, se había explorado el 40.17% del espacio de claves a una tasa final de 5.08 Gclaves/segundo. En esta misma fecha se comenzó el desafío RC5-64, para el 13 de noviembre, a una tasa de 5Gclaves/segundo se esperaba terminar de explorar el espacio de llaves en 104.5 años, para el 9 de septiembre, la cantidad de información transferida entre los clientes y el servidor de claves era de la que la conexión de 128K con a que contaba podía manejar.

El 9 de enero de 1998 se había completado el 0.30% a una tasa de 11.3 Gclaves/segundo y el tiempo estimado para terminar la exploración del espacio de llaves era de 50.6 años. El 13 de enero comenzó el desafío DES-II-1 y se concluyó el 23 de febrero con un 88.4% del espacio de claves explorado a una tasa de 28.1 Gclaves/segundo. Para el 8 de junio se había probado el 1.18% de las claves del desafío RC5-64 y se esperaba explorar el total de las claves del desafío en 29.5 años. De nuevo, el 9 de septiembre del mismo año, e servidor de claves requería de un ancho de banda mayor al que una conexión T1 de 1.544 Mbps proveía.

El 18 de enero de 1999 se inició el desafío DES III y fue resuelto por distributed.net y la Electronic Frontier Foundation en un record de 22 horas, 15 minutos y 4 segundos. [34] [35] [36] [37]

1.6.3 FOLDING@HOME

Folding@Home es un proyecto de la Universidad de Stanford, lanzado en el año 2000, cuya finalidad es la investigación de enfermedades, el diseño de nuevos medicamentos y otros tipos de mecánica molecular analizando los mecanismos a través de los cuales las proteínas realizan un proceso que se llama plegamiento, y las razones por las cuales las proteínas no llegan a plegarse o se despliegan, en la figura 1.15 se muestra el protector de pantalla provisto por Folding@home para las computadoras cliente, en el cual se describe la unidad de trabajo y se muestra la molécula de una proteína. Enfermedades como el Alzheimer, el Parkinson y la enfermedad de las “vacas locas” o BSE (Bovine Spongiform Encephalopaty). Cuando las proteínas no están plegadas interfieren con las funciones normales de las células.

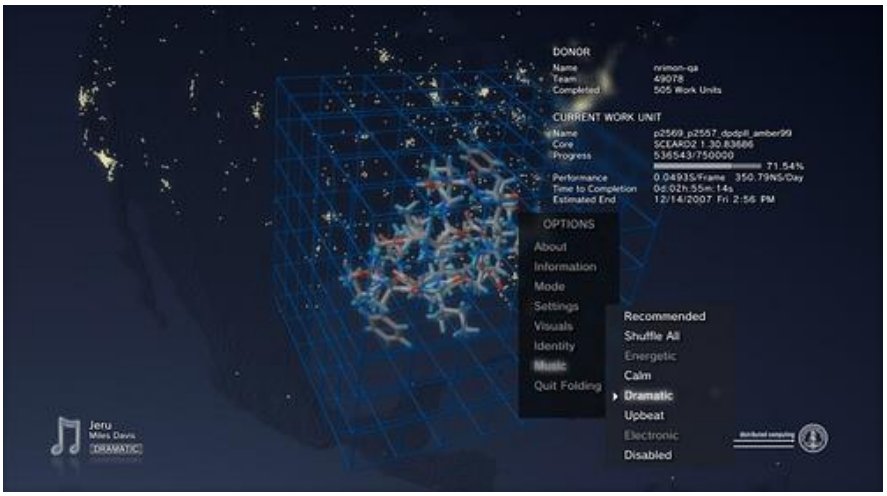


Figura 1.15 Protector de pantalla

Estos fenómenos se analizan a través de simulaciones, que requieren de una gran cantidad de poder de cómputo, debido a que la dinámica de proteínas más importante requiere de simulaciones en la escala de milisegundos o segundos, y una computadora de escritorio o un procesador Cell solo puede computar como máximo 500 nanosegundos de simulación, por lo que se decidió implementar un sistema de cómputo distribuido voluntario que permitiera obtener miles de simulaciones simultaneas.

Este proyecto fue el primero en desarrollar un software que permitiera hacer uso de las GPU's, y en implementar su código en sistemas multiprocesamiento a través de librerías de interface de paso de mensajes, además por un convenio con SONY, en los inicios del proyecto, también hizo uso de librerías especiales para

trasladar su código a clústeres PlayStation3, en la figura 1.16 se muestra un clúster de PS3, y teléfonos inteligentes SONY XPERIA.

Figura 1.16 Clúster de PS3

El software es una implementación del modelo estadístico de Markov, desarrollado y administrado por el equipo del Dr. Vijay Pande, en el Laboratorio Pande de la Universidad de Standford, los componentes del proyecto son unidades de trabajo, las aplicaciones de trabajo denominadas "cores" y el software cliente que se encarga del manejo de las unidades de trabajo y de la comunicación con el servidor. Las unidades de trabajo son fragmentos de simulación de estados del modelo de Markov. Los "cores" son las aplicaciones de trabajo son programas basados en GROMACS, un software especial para la simulación de dinámica de proteínas.

El software actualmente está soportado por los sistemas operativos Windows, OS X y Linux, y está configurado para ejecutarse en segundo plano a una prioridad muy baja por lo que no tiene un gran impacto en el desempeño general de las maquinas cliente. En el año 2014, se creó un cliente que se ejecuta desde los navegadores Google Chrome y Chromium, empleando el cliente nativo de Google. Y en el año 2015 se creó un cliente para la plataforma Android; actualmente el proyecto cuenta con 102,570 equipos voluntarios generando 90,182 TERAFLOPS de procesamiento. [38]

1.6.3.1 ARQUITECTURA

En principio, el cliente instalado en una computadora voluntaria se comunica con un servidor de asignación, el cual le asignará al cliente un servidor de trabajo. Una vez asignado el servidor de trabajo el cliente se comunica con él para solicitarle

una unidad de trabajo, dependiendo de las características de la computadora voluntaria se le asignará una unidad de trabajo, y dependiendo de esa unidad de trabajo, se designará el "core" que descargará el cliente para ejecutar la tarea; cuando el cliente termina de realizar la tarea, envía los resultados al mismo servidor de trabajo o los envía a un servidor especial de recolección de trabajos en caso de que el servidor de trabajo no esté disponible. Por último, se asignan créditos al cliente por el trabajo realizado. [38]

1.6.4 SETI@HOME

SETI es un área de investigación científica cuyo propósito es detectar vida inteligente fuera del planeta Tierra. Existe un enfoque conocido como radio SETI, el cual consiste en buscar señales de radio de banda estrecha empleando radio telescopios; pues debido a que dichas señales no se producen de forma natural, el descubrimiento una señal de este tipo sería evidencia de la existencia de vida extraterrestre. [39]

1.6.4.1 HISTORIA DE SETI

Dos años después del lanzamiento del Sputnik, los físicos Philip Morrison y Giuseppe Cocconi de la Universidad de Cornell publicaron un artículo en el periódico Nature en el que describen la facilidad con la que pueden enviarse mensajes de radio al espacio exterior, a partir de este hecho se propuso la hipótesis de que posiblemente podría detectarse vida extraterrestre utilizando radiotelescopios. Morrison y Cocconi también proponían que la parte correspondiente a la región de microondas del espectro electromagnético era particularmente efectivo para la comunicación a través del espacio. Además, esta parte del espectro es la más tranquila y presenta varias líneas de emisión que podrían ser estudiadas por radio astrónomos de sociedades avanzadas.

Al mismo tiempo en West Virginia el radio astrónomo Frank Drake, en 1960 realizó un experimento en busca de señales de vida extraterrestre; el 8 de abril, empleando un radio telescopio de 26 metros del Observatorio Nacional de Radio Astronomía en Green Bank. Drake apuntó el telescopio hacia dos estrellas cercanas, y realizó un barrido de arriba abajo del espectro de microondas con un receptor de un solo canal. El proyecto fue conocido como Proyecto Ozma, y fue la primera búsqueda SETI moderna. Desde entonces han existido más de 98 proyectos SETI alrededor del mundo. [40]

Figura 1.17 Logotipo del instituto SETI

En 1970 la NASA comenzó a interesarse en la creación de un programa SETI. En 1984 se fundó el Instituto SETI en California, el logotipo del Instituto SETI se muestra en la figura 1.17, con el propósito de facilitar recursos científicos y programas educativos relacionados con la vida en el universo, además de buscar financiamiento privado para una parte del programa SETI de la NASA. En este instituto se creó el Proyecto Phoenix, que realizó 2600 horas de observaciones utilizando un radiotelescopio Parkes de 64 metros ubicado en New South Wales en Australia. [40]

1.6.4.2 EL PROPÓSITO DE SETI@HOME

Las señales obtenidas con los radiotelescopios están compuestas en su mayoría por ruido y señales generadas por el hombre. Los proyectos SETI más recientes analizan las señales digitalmente. El análisis consta de tres fases:

1. Procesar la potencia del espectro que varía con el tiempo.

2. Encontrar una señal candidato por medio de patrones de reconocimiento de potencia del espectro.

3. Descartar señales hechas por el hombre o generadas de forma natural. [41]

Cuando se realiza una búsqueda de señales de banda estrecha, se requiere emplear un canal estrecho alrededor de una frecuencia específica, la amplitud del canal determinará la sensibilidad del sistema. La razón por la que es necesario analizar bandas estrechas se debe a un fenómeno llamado desvío de Doppler.

Este efecto se debe a las aceleraciones del transmisor y el receptor. Debido al movimiento de rotación de la Tierra, un transmisor que escucha a una frecuencia de 1.4 GHz presenta un desvío de 0.16 Hz/s, conociendo este desvío, se pueden hacer las correcciones necesarias para recibir la señal con la mayor resolución

posible, sin embargo, tratándose de una señal de origen desconocido, proveniente de un planeta desconocido, es imposible intentar corregir la señal, por lo que la mejor forma de descubrir una señal de banda estrecha es empleando canales muy estrechos para el análisis, menores a 1 Hz [42]

Una gran cantidad de poder de cómputo destinada a este análisis permite cubrir rangos de frecuencia más grandes y con mayor precisión.

Usualmente el análisis de las señales se realiza en supercomputadoras que se encuentran en las mismas instalaciones en las que se encuentran los radiotelescopios. En 1995 surgió la idea de emplear un gran conjunto de computadoras conectadas a Internet, para aprovechar su poder de cómputo, por lo que David Gedye comenzó el proyecto SETI@home [], el proyecto SETI@HOME provee a los usuarios un protector de pantalla que les permite ver el progreso del análisis que se está llevando a cabo en sus dispositivos, así como a cerca información del archivo de entrada recibido, el cual se muestra en la figura 1.18.[42]

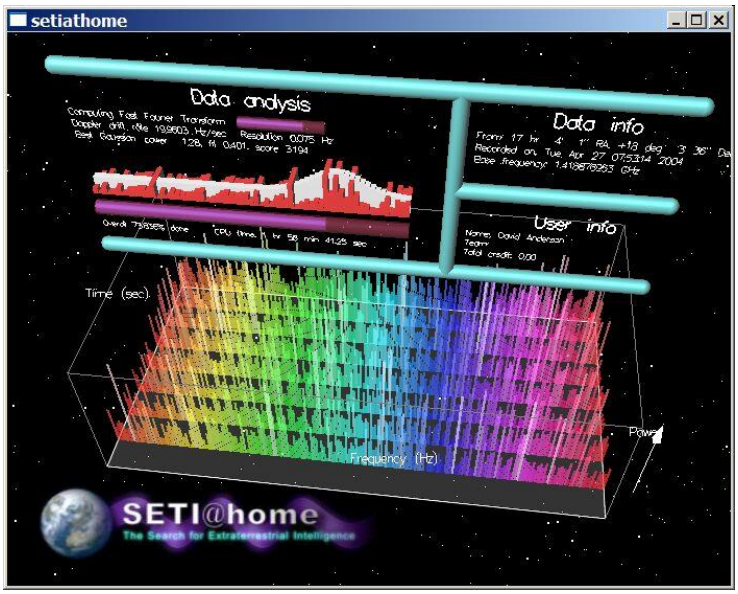


Figura 1.18 Protector de pantalla del proyecto SETI@HOME

SETI@home obtiene las señales del radiotelescopio de Arecibo en Puerto Rico, que se muestra en la figura 1.19, se eligió debido a que es el radiotelescopio más grande y con mayor sensibilidad del mundo, aunque no fue posible obtener permisos para de uso exclusivo para el proyecto SETI@home, gracias al proyecto SERENDIP de la Universidad de Berkeley, se hace uso de una segunda antena, en el mismo radiotelescopio. Mientras la antena principal se fija a un punto determinado en el firmamento, la segunda antena realiza un movimiento en firma

de arco que eventualmente cubre todo el campo visible para el radiotelescopio. Se realizó un arreglo para que el proyecto SETI@home compartiera la información que se obtenía para el proyecto SERENDIP.

Figura 1.19 Radiotelescopio en Arecibo, Puerto Rico

En 1997 cuando surgió el proyecto SETI@home, la conexión a Internet de Arecibo era de 56kbps, por lo que era imposible distribuir la información desde ahí. Debido a este inconveniente, se optó por grabar la información en cintas magnéticas, de 35 GB que eran enviadas por correo a la Universidad de Berkeley. [42]

1.7 COMPUTACIÓN GRID

Consiste en nodos geográficamente dispersos, heterogéneos y débilmente acoplados conectados a través de una red que interactúan para realizar tareas grandes; generalmente se utiliza para cómputo científico que requiere una gran capacidad de cómputo, la computación grid aprovecha el poder de procesamiento no utilizado de los nodos y realizan tareas en tiempos cortos. El tamaño de una grid varía desde una red pequeña en una empresa hasta una red compuesta por miles de nodos distribuidos por todo el mundo. La coordinación entre nodos se realiza a través de librerías middleware. [25]

CAPITULO 2: ANÁLISIS DE LAS PLATAFORMAS DE CÓMPUTO DISTRIBUIDO

El middleware es el software que se encarga de conectar los componentes de software que se ejecutan en un sistema distribuido, se encarga de proveer la transparencia, concurrencia y seguridad. Es una capa de software que se ejecuta sobre los sistemas operativos y los protocolos de comunicación, a continuación, se explican de forma más detallada las funciones del middleware:

- ✓ Provee transparencia: Se encarga de ocultar la naturaleza distribuida del sistema, ya que un sistema distribuido consta de varios elementos interconectados, también se encarga de ocultar la heterogeneidad de dichos componentes.

- ✓ Provee interfaces estándar de alto nivel para los desarrolladores de aplicaciones e integradores, de tal forma que las aplicaciones desarrolladas sean portables, reutilizables y que pueda interoperar.

- ✓ Provee un conjunto de servicios comunes que realizan funciones de propósito general, de tal forma que se evite duplicar operaciones y además sea más fácil la colaboración entre aplicaciones. [43]

2.1 BOINC

Berkeley Open Infrastructure for Network Computing (BOINC) es una plataforma de software para computación voluntaria y en Grid, que fue originalmente creada para el proyecto SETI@Home, actualmente sirve de plataforma para otros proyectos científicos de computación distribuida que requieren grandes cantidades de espacio de almacenamiento o de memoria.

Figura 2.1 Logotipo de BOINC

BOINC aprovecha el tiempo en desuso de las computadoras voluntarias para realizar cómputo científico. El software de servidor de BOINC es altamente eficiente, y puede manejar millones de trabajos al día. El software de cliente de BOINC puede ejecutarse en múltiples plataformas de hardware y software (Linux, Windows, Mac OS, Android, etc). Actualmente existen 598,597 hosts activos. Los voluntarios pueden participar en múltiples proyectos, y pueden controlar como se distribuirán sus recursos entre los diferentes proyectos, actualmente existen 271,219 de voluntarios activos. [44]

En la tabla 2.1 se presentan las características de la plataforma BOINC:

Tabla 2.1 Características de la plataforma BOINC [45]

Características de BOINC	
Autonomía de los proyectos.	Actualmente muchos proyectos utilizan BOINC, cada proyecto es independiente y opera en sus propios servidores y bases de datos. No existe un directorio central o un proceso de aprobación.
Flexibilidad para los voluntarios.	Los voluntarios pueden participar en múltiples proyectos, además pueden elegir en que proyectos participar y como se dividirán los recursos entre los proyectos que elijan.
Infraestructura de aplicación flexible.	Aplicaciones que ya están programadas en los lenguajes C, C++ y Fortran pueden ejecutarse como aplicaciones de BOINC, haciendo pequeñas modificaciones en el código o a veces sin necesidad de modificarlas. Una aplicación puede constar de varios archivos (varios programas y un script de coordinación).
Escalabilidad y desempeño de los servidores.	El software del servidor de BOINC es extremadamente eficiente, de tal forma que un servidor de gama media puede enviar y manejar millones de trabajos por día. La arquitectura del servidor es altamente escalable, lo que hace más fácil incrementar la capacidad del servidor o la disponibilidad añadiendo más máquinas.

Disponibilidad del código fuente.	BOINC se distribuye bajo la licencia *Lesser General Public License*. Las aplicaciones BOINC no necesariamente son *open source*.
Soporte para grandes cantidades de datos.	BOINC soporta aplicaciones que producen o consumen grandes cantidades de datos, o que utilizan grandes cantidades de memoria. La distribución y recolección de datos puede distribuirse entre varios servidores, y las maquinas participantes transfieren cantidades grandes de datos de forma discreta. Los usuarios pueden limitar el espacio en disco y el ancho de banda que se dedicara trabajar con los proyectos. Solo se envía trabajo a los clientes que son capaces de manejarlo.
Múltiples plataformas participantes.	El cliente BOINC está disponible para la mayoría de las plataformas (Mac OS X, Windows, Linux y otros sistemas Unix). El cliente puede manejar múltiples CPUs.
Arquitectura de software abierta y extensible.	BOINC provee interfaces documentadas para varios de sus componentes clave, lo que hace posible que otros desarrolladores creen software y sitios web que extiendan BOINC.
Características comunitarias para voluntarios.	BOINC provee herramientas web como tableros de mensajes, perfiles de usuario, mensajería privada, que alienta a los voluntarios a formar comunidades online.

2.1.1 APLICACIONES APTAS PARA BOINC.

BOINC está diseñado para soportar aplicaciones que tienen grandes requerimientos de computación, de almacenamiento o ambos. El requerimiento principal de la aplicación es que sea divisible entre un gran número de trabajos que puedan realizarse independientemente.

Si el proyecto utilizará recursos voluntarios, una aplicación debe ser interesante para poder obtener un gran número de participantes, un proyecto debe tener

recursos y compromiso para mantener el interés del público generando gráficos interesantes en la aplicación o creando un sitio web atractivo. Debido a que los datos de salida y de entrada se envían a través de conexiones de internet comerciales, que pueden ser caras y lentas, por lo cual el cómputo voluntario no es una buena opción si la aplicación genera o consume más de un gigabyte de datos por día.

Las mejores aplicaciones para BOINC son aquellas que trabajan con “paquetes de tareas”, es decir, grandes conjuntos de trabajos independientes, con requerimientos modestos de memoria y almacenaje. Comúnmente las aplicaciones de BOINC son aquellas que especializadas en:

- ✓ Simulaciones de sistemas físicos.
- ✓ Computo intensivo para el análisis de grandes conjuntos de datos.
- ✓ Exploración de grandes espacios de búsqueda (incluyendo algoritmos genéticos).

No es necesario reescribir las aplicaciones, incluso se pueden emplear los ejecutables sin código fuente.

BOINC funciona bajo el modelo despachador-trabajador, existe un servidor que se encarga de enviar tareas a los clientes y de recibir, validar y almacenar los resultados; mientras que los clientes reciben las tareas, realizan los cálculos, envían los resultados al servidor, y solicitan mas tareas, lo anterior se ilustra en la figura 2.2. [46]

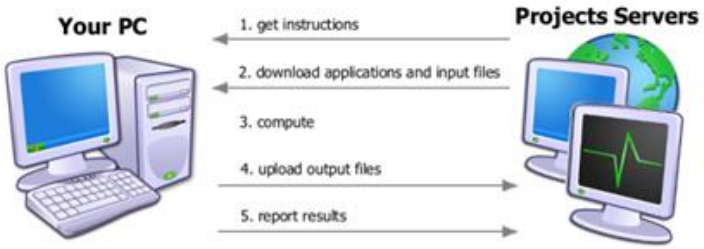


Figura 2.2 Funcionamiento de BOINC

En el anexo A, se muestran los diagramas de flujo del comportamiento del cliente y el servidor.

2.2 PLANETLAB

Es una red de investigación global que promueve el desarrollo de nuevos servicios de red. Desde sus inicios en 2003, más de 1000 investigadores de instituciones académicas y laboratorios de investigación industrial utilizan PlanetLab para desarrollar nuevas tecnologías de almacenamiento distribuido, mapeo de redes, sistemas peer-to-peer, tablas hash distribuidas, y procesamiento de consultas, el logotipo de PlanetLab se muestra en la figura 2.3.

Figura 2.3 Logotipo de PlanetLab

Actualmente consta de 1337 nodos y 681 sitios. La distribución de los las computadoras cliente que participan en proyectos de PlanetLab se muestra en la figura 2.4.

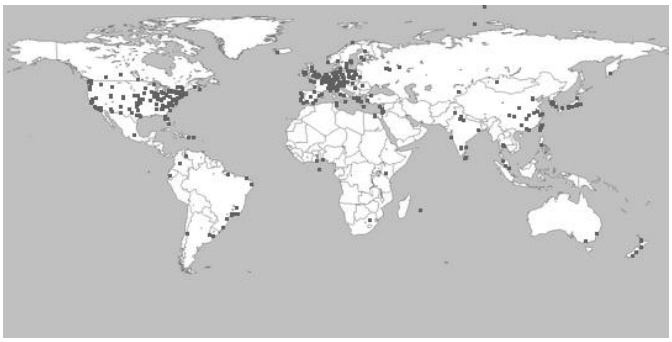


Figura 2.4 Distribución de nodos de PlanetLab

Fue creado por Larry Peterson (Princeton) y David Culler (UC Berkeley e Intel Research). Actualmente el consorcio de PlanetLab está formado por personas de las varias instituciones las cuales se muestran en la tabla 2.2:

Tabla 2.2 Consorcio PlanetLab [47]

PlanetLab
Universidad de Princeton
UC Berkeley
Intel
Universidad de Washington

Universidad de Cambridge
UPMC
HP Labs
ETH Zürich
Google
AT&T
France Telecom
AT Corporation
DoCoMo Communications Laboratories USA
Lucent - Bell Labs
NEC Laboratories
Telecom Italia.

2.3 IBM GRIDTOOLBOX (GLOBUS)

Es software de código abierto cuyo desarrollo esta lidereado por Argonne National Laboratory, de la Universidad del Sur de California y la Universidad de Chicago. Este sistema consta de tres componentes principales:

- Administración de Recursos
- Servicios de Información
- Administración de Datos

Las tecnologías empleadas para realizar estas tareas son:

Grid ResourceAllocation Management (GRAM), Monitoring and Discovery Service (MDS), y Grid File Transfer Protocol (GridFTP). Además, todos estos componentes utilizan el protocolode seguridad Grid Security Infraestructure (GSI) en la capa de conexión. [48]

2.4 HTCONDOR

Es un sistema de administración de carga de trabajo para trabajos de cómputo intensivo, esta plataforma es de código abierto y es desarrollada por la Universidad de Wisconsin-Madison desde hace 15 años. Al igual que otros sistemas, HTCondor provee un mecanismo de encolamiento de trabajo, una política de programación, un esquema de prioridades, monitoreo de recursos, y administración de recursos su logotipo se muestra en la figura 2.5.

Figura 2.5 Logotipo de HTCondor

Los usuarios agregan trabajos secuenciales o paralelos a HTCondor, y estos son encolados, escoge cuando y donde ejecutar los trabajos de acuerdo a una política, monitorea el progreso de los trabajos e informa al usuario cuando son completados. Utiliza el tiempo ocioso de las estaciones de trabajo para realizar los trabajos. [49]

2.5 XTREME WEB

Es una plataforma de cómputo distribuido voluntario cuyo objetivo es estudiar el comportamiento de los sistemas de cómputo distribuido voluntario de forma experimental. Surgió a petición del Observatorio Pierre Auger, debido a que cada año requerían hacer una simulación en $6x10^5$ diferentes archivos de entrada lo equivalente a $6x10^5$ horas de trabajo de PC que operan a 300Mhz.

Xtreme Web no contempla el tamaño de los componentes del entorno de acuerdo con las especificaciones del proyecto, no contempla alto desempeño ni manejo de la carga de archivos de entrada para los algoritmos de calendarización. Para que un cliente descargue la aplicación de trabajo se sigue un proceso de comunicación ad-hoc.

Solo instituciones de confianza pueden proponer códigos que se puedan integrar a Xtreme Web, bajo el siguiente procedimiento.

- ✓ El código se prueba en clientes dedicados.
- ✓ El código esta encriptado antes de que lo descarguen los clientes.
- ✓ El procedimiento para la descarga del código emplea una clave privada para asegurar la transacción.

La comunicación inicia del lado del cliente, enviando peticiones de tareas de esta manera se evitan problemas con el firewall. El protocolo entre los clientes y los servidores es independiente de la capa de comunicación, puede implementarse mediante sockets TCP-UDP/IP o a través de una capa de más alto nivel como CORBA o Java RMI. En la figura 2.6 se muestra la estructura extendida de un sistema distribuido que emplea XtremeWeb.

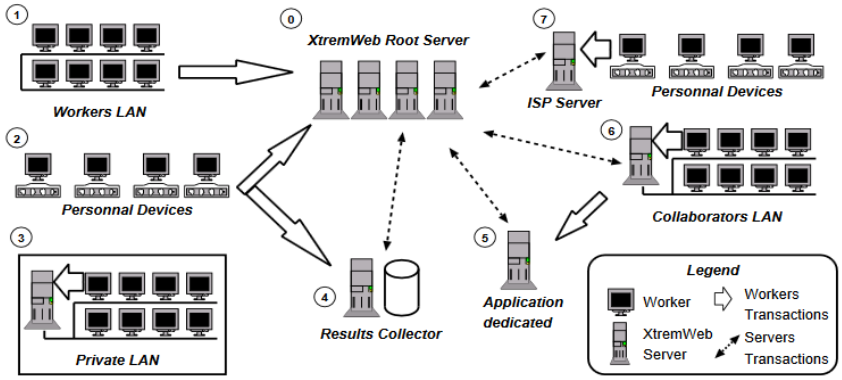


Figura 2.6 Estructura de un sistema distribuido empleando XtremeWeb

Esta implementación también permite utilizar protocolos más especializados tales como SSL y RSVP. La plataforma es lo suficientemente flexible para dedicar servidores a tareas específicas como recolectar resultados. [32]

CAPITULO 3: CONFIGURACIÓN DEL ENTORNO DE DESARROLLO PARA PROYECTOS BOINC

Una vez que se conocen las distintas características de las plataformas disponibles de cómputo distribuido, la opción más viable y flexible para trabajar es el middleware BOINC, debido a que ofrece una plataforma completa lo suficientemente robusta para realizar proyectos que requieren una gran cantidad de procesamiento. En primera instancia, se requiere la implementación de un servidor de proyectos. De acuerdo con las especificaciones, dicho servidor debe emplear un sistema operativo GNU/Linux de 64 bits, las características de memoria RAM y capacidad de disco duro dependen del proyecto o proyectos que alojará dicho servidor. [50]

3.1 IMPLEMENTACIÓN DE UN SERVIDOR BOINC

En la presente sección se realizará la implementación de un servidor BOINC, además, en este servidor también servirá para el desarrollo de aplicaciones para BOINC por lo que se compilará el software del servidor, el API de BOINC y el cliente, con el fin de realizar todas las pruebas necesarias antes de lanzar un proyecto. El servidor se implementará sobre una distribución de GNU/Linux Debian 8 de 64 bits, en el momento de la instalación se crea el usuario “boincadm”.

3.1.2 CONFIGURACIÓN

Las dependencias necesarias para compilas el servidor, el API, y el cliente son las siguientes:

git, make, m4, automake, libtool, autoconf, pkg-config, apache2, php5, php5-gd, php5-cli, php5-mysql, mysql-server, python-mysqldb, libmysql++--dev, libssl-dev, dh-autoreconf, libapache2-mod-php5, libmysqlclient-dev, gcc, g++, openssl, libcurl4-openssl-dev, libxss-dev, libnotify-dev, libxcb-util0-dev, libsqlite3-dev [51]

Se descarga el código fuente de BOINC con el siguiente comando [50]:

```
$ git clone https://github.com/BOINC/boinc boinc
```

3.1.3 CONFIGURACIÓN DEL SERVIDOR LAMP DE BOINC

El servidor BOINC en esencia es un servidor LAMP, una vez instalados los componentes del servidor se continúa la configuración de MySQL y Apache.

3.1.3.1 CONFIGURACIÓN DE APACHE

Lo primero es verificar que el servidor Apache éste funcionando correctamente, para esto solo se debe abrir un navegador web y escribir en la barra de direcciones la palabra localhost, si está correctamente configurado se abrirá la página de bienvenida de Apache. La página de bienvenida puede cambiar dependiendo de la distribución de GNU/Linux empleada, por ejemplo en la figura 3.1 se muestra la página de bienvenida en un servidor Fedora.

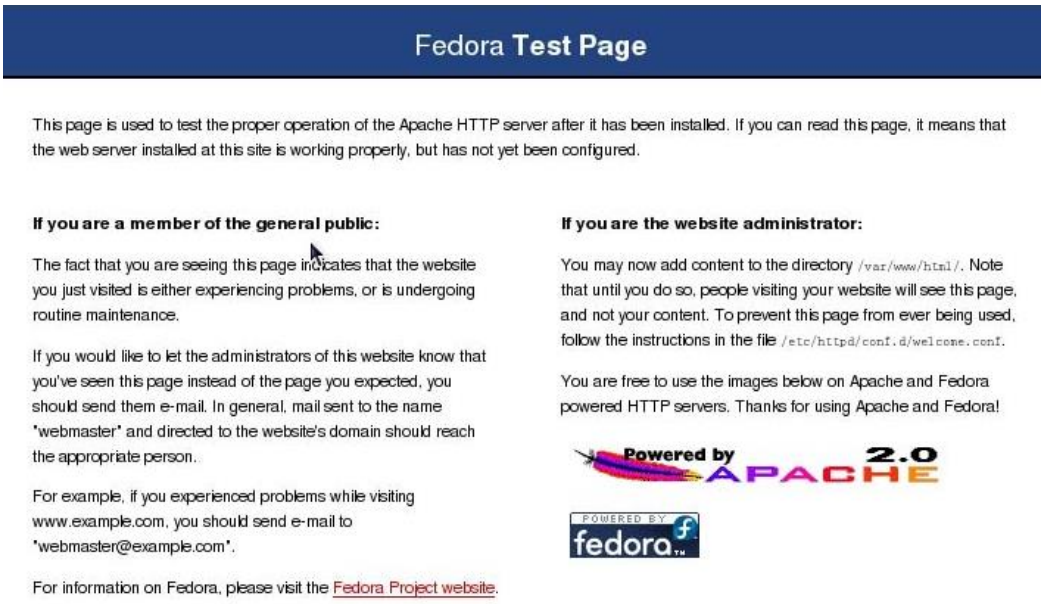


Figura 3.1 Página de bienvenida en un servidor Fedora.

Para un servidor Debian, la página de bienvenida se verá como la de la figura 3.2.

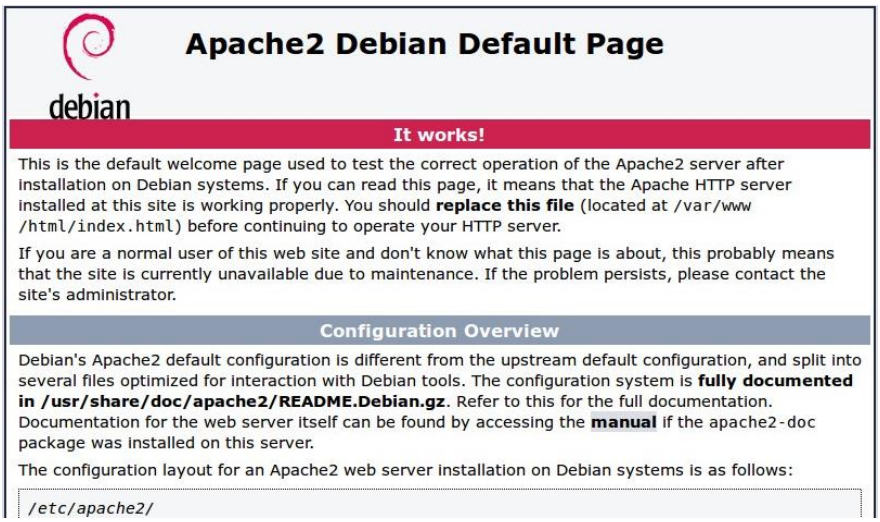


Figura 3.2 Página de bienvenida en un servidor Debian.

En Apache 2.4, el módulo CGI, esta deshabilitado por defecto, por lo que hay que habilitarlo manualmente con este comando como superusuario:

```
# a2enmod cgi
```

Cuando se instala Apache se crea también un usuario que se encarga de manipular los directorios, archivos y aplicaciones del servidor, esto genera problemas con BOINC, ya que los directorios que le pertenecen al usuario Apache solo pueden ser modificados por este usuario, para solucionar este problema se debe agregar el usuario creado por apache al grupo boincadm[50], en Debian el usuario es www-data:

```
# usermod -aG boincadm www-data
```

Y en Fedora el usuario es httpd

```
#usermod -aG boincadm httpd
```

3.1.3.2 CONFIGURACIÓN DE MYSQL

Al momento de la instalación, es posible que se solicite la creación de una contraseña para el usuario root, una vez instalado MySQL se termina la configuración con el comando:

```
# mysql_secure_installation
```

Si no se configuró una contraseña para el usuario root al momento de la instalación, al ejecutar este comando se puede configurar una, o si se creó una contraseña para el usuario root, esta se puede cambiar por una más segura, además, este comando eliminará los usuarios y bases de datos de ejemplo [52].

Una vez configurado MySQL se le dan permisos al usuario “boincadm” con los siguientes comandos como usuario normal:

```
$ mysql -u root -p
```

Se proporciona la contraseña creada para el usuario root y dentro de MySQL se crean los permisos y se configura una contraseña en blanco para el usuario boincadm[50]:

```
mysql > GRANT ALL ON *.* TO 'boincadm'@'localhost';
mysql > SET PASSWORD FOR 'boincadm'@'localhost' = '';
```

3.1.3.3 CONFIGURACIÓN DE APACHE PARA PHP

El último paso es configurar Apache para que busque y ejecute primero los archivos index.php, ya que por defecto los primeros archivos que toma en cuenta son los index.html, esto se hace modificando el archivo /etc/apache2/mods-enabled/dir.conf, en la opción DirectoryIndex de la configuración debe aparecer primero index.php, la configuración debe verse como sigue:

```
<IfModule mod_dir.c>
     DirectoryIndex index.php index.html index.cgi index.pl
     index.xhtml index.htm
</IfModule>
```

Cada que se modifica uno de los archivos de configuración de Apache se debe reiniciar el servicio para que los cambios surtan efecto, esta acción se ilustra en la figura 3.3.[52]

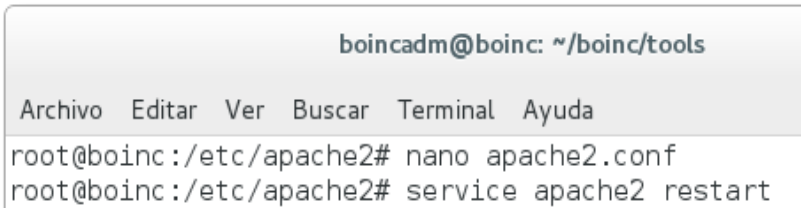


Figura 3.3 Reinicio del servicio Apache2

3.1.3.4 COMPILACIÓN DEL CÓDIGO FUENTE

Una vez descargado el código fuente, en el directorio de descarga, en este caso llamado “boinc” se ejecutan los siguientes comandos:

```
$ ./_autosetup
```

En la figura 3.4 se muestra parte de la salida de la ejecución de _autosetup, si no se han cubierto los requerimientos para la creación del servidor, el programa mostrará la lista de las dependencias faltantes, de acuerdo a la configuración elegida.

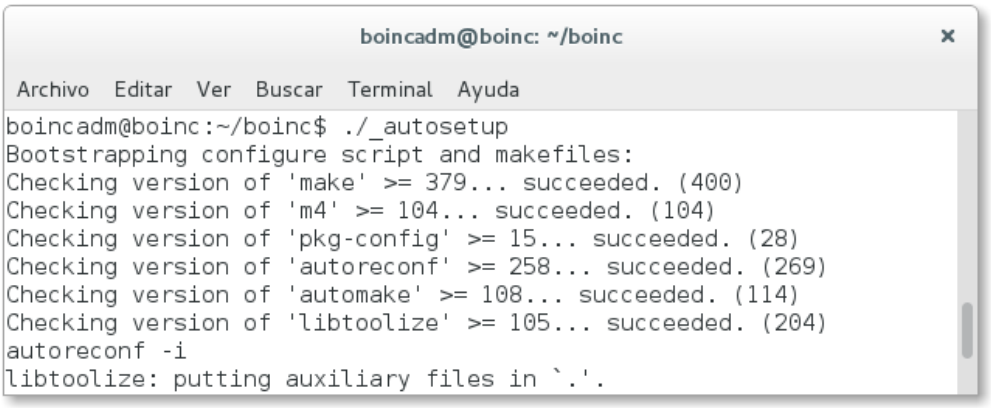


Figura 3.4 Ejecución del script _autosetup

Debido a que no se necesita el administrador para las pruebas se puede deshabilitar su compilación, en la figura 3.5 se muestra la salida del comando configure.

```
$ ./configure --disable-manager
```

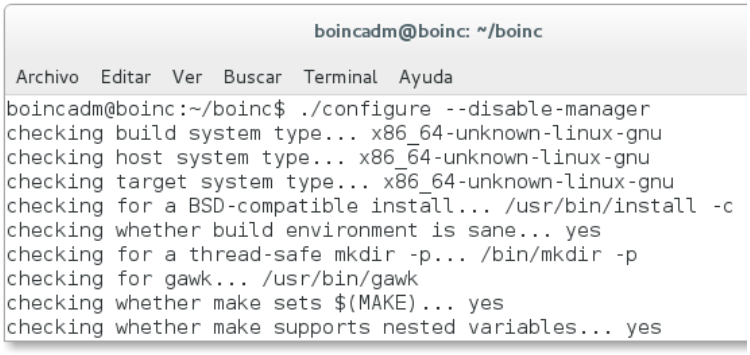


Figura 3.5 Ejecución del script configure

Una vez configurado el software, se genera un Makefile, el ultimo paso es compilar el software con el comando 'make'[53]. La figura 3.6 muestra la salida del comando ‘make’.

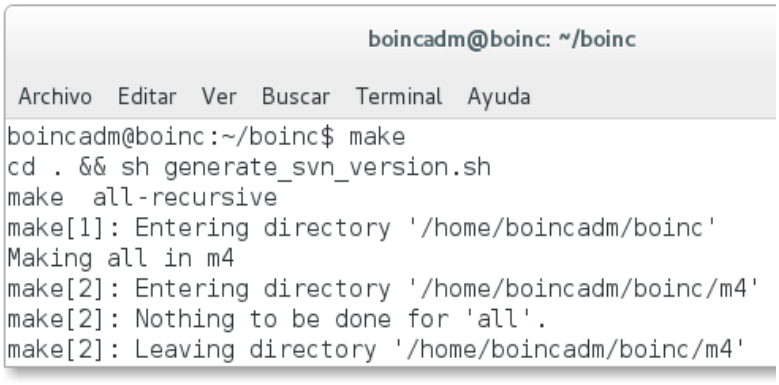


Figura 3.6 Ejecución del comando make

3.1.3.5 PRUEBA DEL SERVIDOR

Una vez configurado el servidor BOINC se creará un proyecto de prueba, con la configuración actual, cuando se intenta crear un proyecto de prueba aparecerá un error, ya que en la carpeta samples/example_app no se encuentran las plantillas de trabajo de la aplicación, estas plantillas se encuentran en la carpeta tools/ y es necesario copiarlas a la carpeta samples/example_app para que el script de creación de proyectos pueda copiarlas a la carpeta del proyecto.

En el directorio donde se descargó y compiló BOINC, se encuentra la carpeta tools, en la que se encuentra el script de creación de proyectos, para crear un proyecto de prueba se ejecuta de la siguiente forma[54]:

```
$ make_project --test_app test
```

Con este comando se creará un proyecto de prueba llamado test, que contendrá la aplicación de ejemplo que recibe un archivo de texto en minúsculas como entrada y devuelve un archivo de salida con el mismo texto en mayúsculas, en la figura 3.7 se puede ver la ejecución del comando make_project, como puede observarse, se crea la base de datos, se copian los archivos del sitio web del proyecto y los programas daemon, además, se crean los archivos de configuración.

Cuando se crea un proyecto por primera vez, se crea un directorio llamado "projects" que contendrá todos los proyectos que se creen en el servidor.

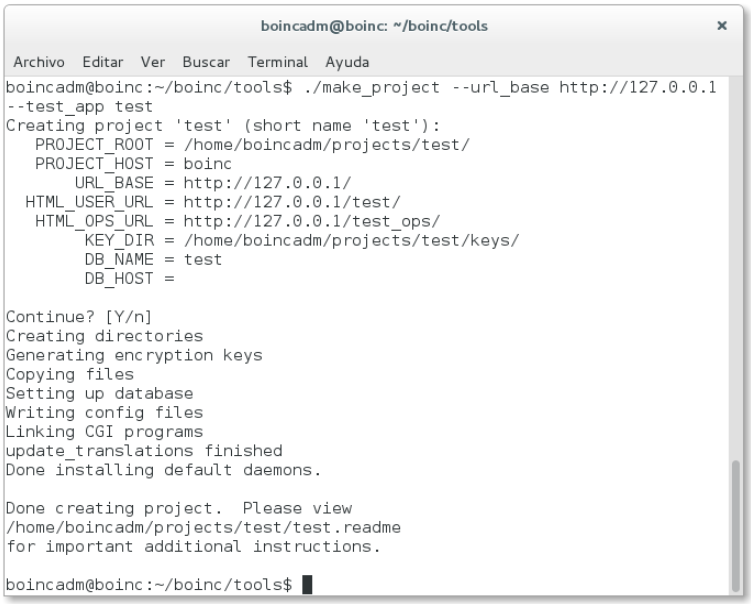


Figura 3.7 Creación de un proyecto de prueba con el script make_project

Para Apache 2.4 se debe agregar la ruta a esta carpeta en el archivo de configuración de Apache /etc/apache2/apache2.conf, como se ilustra en la figura 3.8.

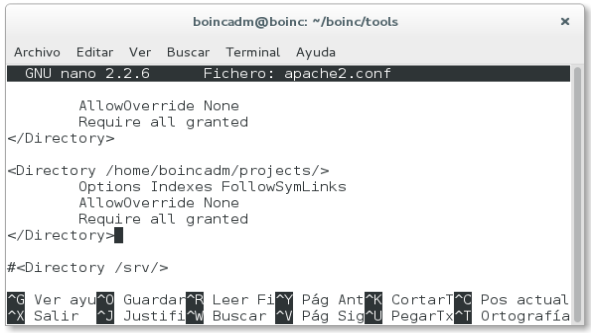


Figura 3.8 Configuración de acceso al directorio "projects"

Además dependiendo del sistema operativo empleado puede ser necesario cambiar los permisos de la carpeta home en la que se almacenan los proyectos, la carpeta home debe tener permisos 711, es decir, permisos de lectura, escritura y ejecución para el propietario, y para cualquier otro usuario permiso de ejecución solamente.

3.1.3.6 CONFIGURACIÓN DEL PROYECTO

A continuación se muestran las configuraciones necesarias para lanzar un proyecto BOINC.

3.1.3.6.1 APACHE

Se debe añadir el proyecto al servidor Apache, al crear un proyecto nuevo, el script make_project crea un archivo de configuración con todas las directivas necesarias para que Apache pueda manejar el proyecto. El archivo se llama nombre_del_proyecto.httpd.conf, para el proyecto de ejemplo llamado “test” el nombre del archivo es test.httpd.conf. Se debe añadir el contenido de este archivo al archivo de configuración de Apache /etc/apache2.conf, o se puede incluir la ruta de este archivo con la directiva Include. Si la configuración fue correcta en un navegador web se podrá visualizar la página web del proyecto como se muestra en la figura 3.9, [54].

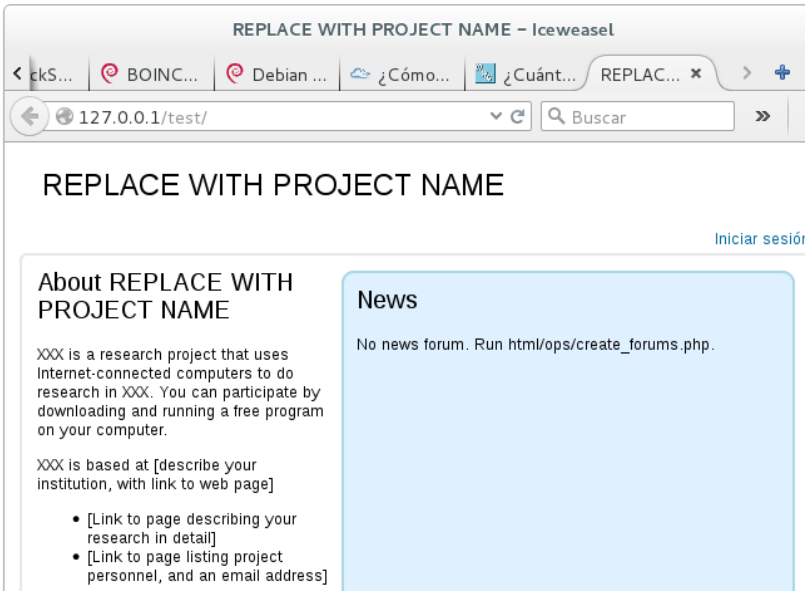


Figura 3.9 Sitio web del proyecto de prueba “test”

Para Apache 2.4 se debe modificar el archivo de configuración del proyecto sustituyendo las directivas “Order”, “Allow” y “Deny” por las directivas “Require all granted” y “Require all denied”, como se muestra en la tabla 3.1[55].

Tabla 3.1 Comparativa de cambios en la configuración de Apache 2.2 y Apache 2.4

# Para Apache 2.2	# Para Apache 2.4
<Directory "/home/boincadm/projects/test/keys"> Order deny,allow Deny from all </Directory>	<Directory "/home/boincadm/projects/test/keys"> Require all denied </Directory>

3.1.3.6.2 CRON

Se debe agregar el servicio a cron, esto se puede hacer de dos formas:

1. Se copia el comando contenido en el archivo test.cronjob y se pega en el archivo de configuración de cron que se edita con el comando crontab -e.
2. Se ejecuta el comando crontab test.cronjob, que copiará automáticamente el contenido del archivo en el archivo de configuración de cron.

Esto hará que el proyecto se actualice cada 5 minutos, la configuración se ilustra en la figura 3.10 [54].

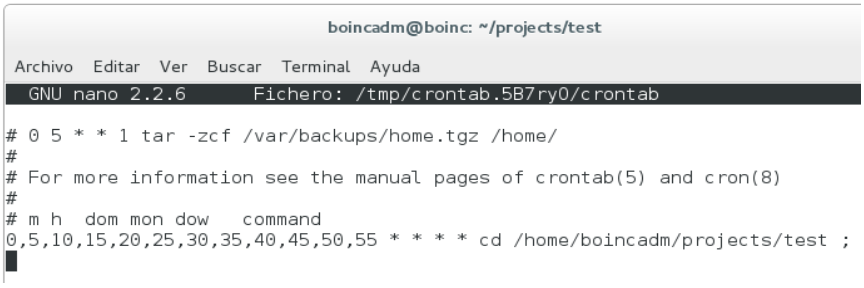


Figura 3.10 Archivo de configuración crontab

3.1.3.6.3 INICIALIZACIÓN DE LA BASE DE DATOS DEL PROYECTO

El siguiente paso es inicializar la base de datos del proyecto y cargar las distintas versiones de las aplicaciones en la base de datos, mediante los comandos xadd y update_versions, que se encuentran en el directorio bin/ del proyecto, el comando

xadd agrega a la base de datos del proyecto las plataformas a las que va dirigida la aplicación, las cuales se encuentran listadas en el archivo project.xml, la salida del comando xadd se muestra en la figura 3.11.

```
$ bin/xadd
```

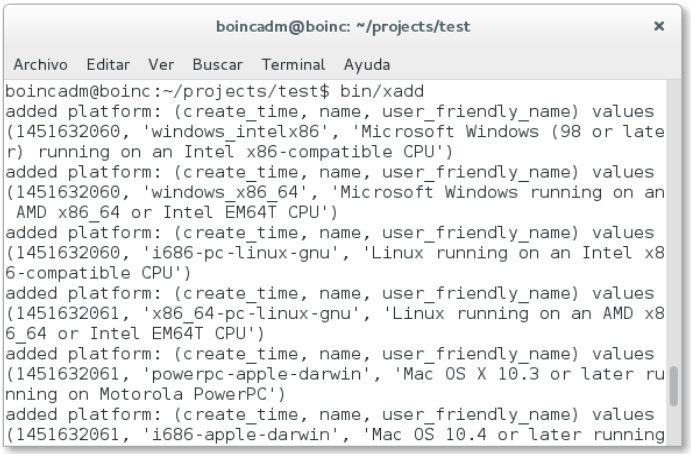


Figura 3.11 Ejecución del comando xadd

Una vez registradas las plataformas, se agregan las aplicaciones de trabajo dirigidas a cada plataforma empleando el script update_versions

```
$ bin/update_versions
```

Al ejecutar el script update_versions se almacena la información de las aplicaciones de trabajo en la base de datos, si no existe la plataforma para la una aplicación de trabajo existente en el servidor, el script enviará un mensaje de error, la salida de el script update_versions se ilustra en la figura 3.12.

Figura 3.12 Ejecución del comando update_versions

3.1.3.6.4 INICIAR EL PROYECTO.

El último paso es iniciar el proyecto, en el directorio del proyecto se ejecuta el comando start, la salida de este comando se miestra en la figura 3.13, y puede observarse que se encarga de levantar los procesos daemon que requiere el proyecto:

```
$ bin/start
```

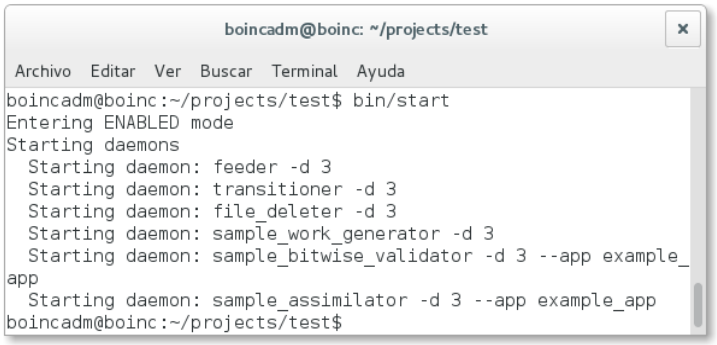


Figura 3.13 Inicialización del proyecto con el comando start

Si se creara un proyecto vacío, habría que crear una estructura de directorios para las diferentes versiones de la aplicación a distribuir con BOINC, además de los ejecutables de la aplicación compilados en las plataformas a las que estará dirigida y sus respectivos archivos de configuración, en este caso, se crea automaticamente la estructura de directorios y se copian las versiones existentes de la aplicación de prueba.[54]

3.1.3.6.5 PRUEBA DEL PROYECTO

Se puede probar el proyecto desde el mismo servidor, creando una carpeta que en este ejemplo se llamará boinc_test, dentro de esta carpeta se copia el cliente de BOINC (boinc_client) y la herramienta de comandos del cliente de BOINC (boinccmd) como se muestra en la figura 3.14.

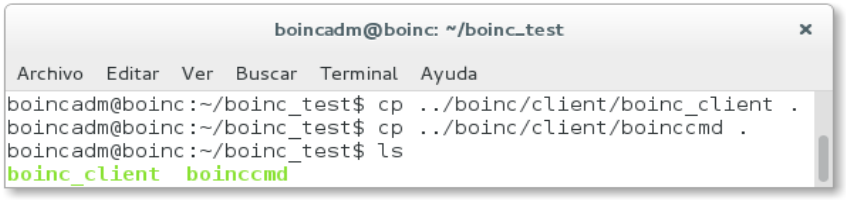


Figura 3.14 Copia del cliente y la herramienta de comandos

Se ejecuta el cliente como cualquier aplicación:

```
$ ./boinc_client
```

En la figura 3.15, se puede ver al cliente en ejecución, vez que se ejecuta el cliente, no se detiene, hasta que el usuario decida terminar el proceso, por lo que es necesario ejecutar los demás comandos desde otra terminal.

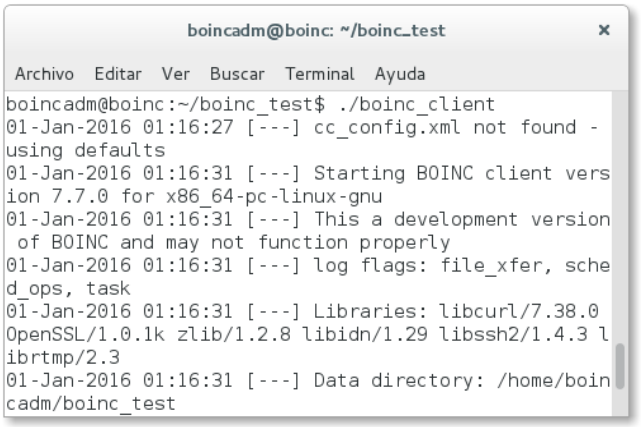


Figura 3.15 Ejecución del cliente de BOINC

Se crea una cuenta en el servidor, ya sea a través de la página web del proyecto o a través de la herramienta de línea de comandos del cliente. Cuando se crea una cuenta, se crea una clave asociada a la cuenta para añadir el proyecto al cliente de BOINC. Esto se puede hacer a través de la herramienta de línea de comandos como se muestra en la figura 3.16:

```
$ boinccmd --project_attach http://127.0.0.1/test/ clave
```

Figura 3.16 Conexión del cliente al proyecto

En donde “clave” es la clave generada al crear la cuenta. El cliente comenzará a realizar los trabajos, y a enviarlos al servidor, si la configuración es correcta, en la terminal en la que se ejecutó el cliente se podrá ver que comienza la descarga y ejecución de los trabajos como se ilustra en la figura 3.17.[54]

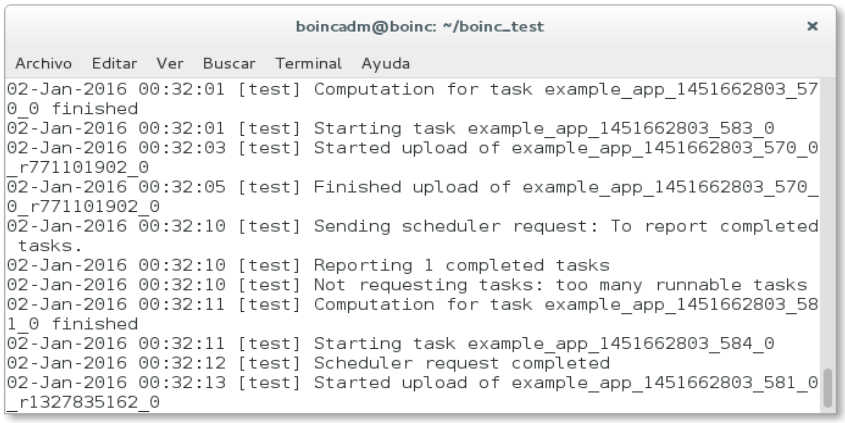


Figura 3.17 Cliente por consola de comandos ejecutando tareas

3.2 COMPONENTES DE UN PROYECTO EMPLEANDO BOINC

Un proyecto es una entidad que realiza cómputo distribuido empleando el middleware BOINC. Cada proyecto es independiente, se identifica por una URL maestra, y cuenta con sus propias aplicaciones, base de datos, sitio web, etc. En un servidor BOINC pueden coexistir varios proyectos.

Un proyecto está compuesto por:

- ✓ Una base de datos en MySQL.
- ✓ Una estructura de directorios.
- ✓ Un archivo de configuración.
- ✓ Un archivo de descripción de plataformas.
- ✓ Una aplicación de trabajo.
- ✓ Un conjunto de programas daemon.

A continuación se presenta una breve descripción de cada uno de los componentes de un proyecto BOINC.

3.2.1 BASE DE DATOS EN MYSQL

Para cada proyecto se crea una base de datos que almacenará la información de las aplicaciones, las unidades de trabajo y la información de los voluntarios y sus equipos de cómputo. Las tablas de las que se compone la base de datos son:

Tabla 3.1 Estructura de la base de datos de un proyecto [56]

Estructura de la base de datos de un proyecto	
Plataform	Los objetivos de compilación del cliente y/o las aplicaciones; es decir, combinación de sistemas operativos y arquitecturas de procesador.
App	Las aplicaciones que se ejecutan del lado del cliente. El núcleo del cliente también es tratado como una aplicación cuyo nombre será 'core_client'.
App_version	Versiones de aplicaciones, es decir, las aplicaciones desarrolladas para cada plataforma. Cada registro incluye una URL para descargar el ejecutable, y el MD5 checksum del ejecutable.
User	Describe a los usuarios que participan en el proyecto, incluyendo su dirección de correo, nombre, contraseña web, y el autenticador que se les asigna al crear una cuenta en el sitio del proyecto.
Host	Describe a los hosts
Workunit	Contiene la descripción de las unidades de trabajo. La descripción de los archivos de entrada se almacena en un archivo XML en un campo blob. También se almacena un conteo del número de resultados ligados a la unidad de trabajo, la cantidad de resultados que han sido enviados, que han tenido éxito y que han fallado.

Result	Almacena la información de los resultados obtenidos. Incluye un campo de 'estado', en el que se especifica si el resultado ha sido procesado. Almacena información que solo es relevante solo cuando el resultado ha sido devuelto: tiempo de CPU, estado de salida y estado de validación

3.2.2 ESTRUCTURA DE DIRECTORIOS

Cuando se crea un proyecto, se crea la estructura de directorios que contendrá los archivos de configuración, las versiones de aplicaciones, los archivos de entrada y de salida y el sitio web del proyecto. En la figura 3.18 se muestra la estructura del directorio de un proyecto en BOINC.

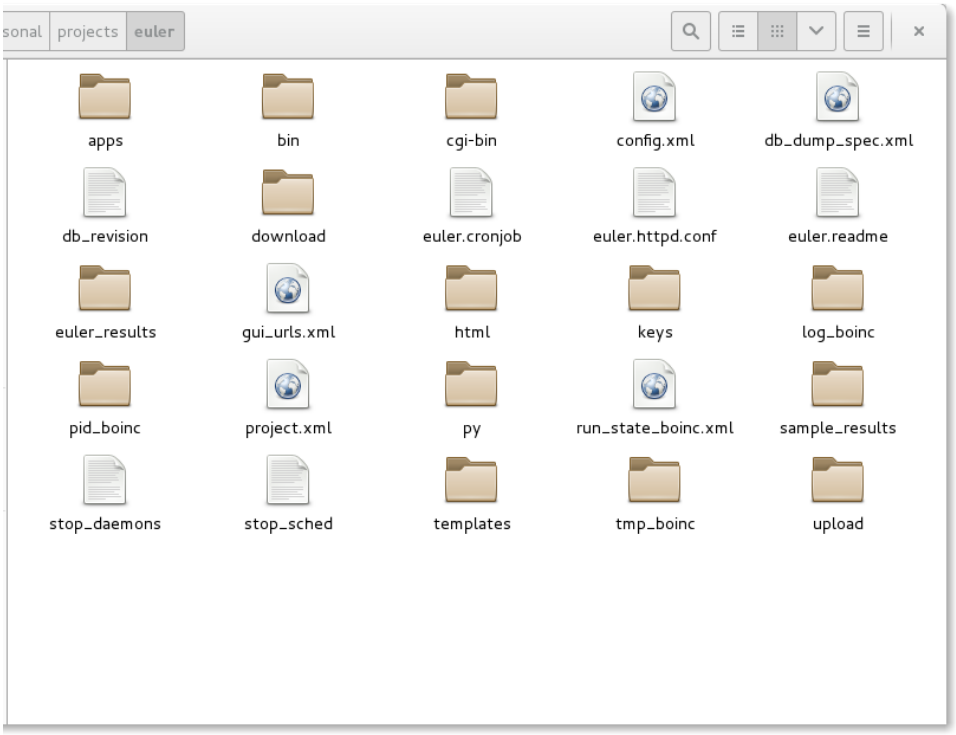


Figura 3.18 Directorios creados del proyecto Euler

La estructura de directorios de cada proyecto se crea en la carpeta projects que se crea automáticamente cuando se crea el primer proyecto en el servidor, la carpeta raíz del proyecto será nombrada igual que el proyecto, en el caso de la imagen, el nombre del proyecto es Euler. Cada directorio de proyecto contiene los directorios descritos en la tabla 3.2.

Tabla 3.2 Descripción del contenido de las carpetas del directorio de trabajo

Descripción del contenido de las carpetas del directorio de trabajo	
Apps	Incluye las versiones de la aplicación de trabajo
Bin	Contiene los scripts y programas daemon
cgi-bin	Contiene programas que utilizan gráficos
Log	Contiene las bitácoras de
Pid	Contiene archivos log y pid del proyecto
Download	Almacena las versiones de las aplicaciones y sus archivos asociados, así como los archivos de entrada de la aplicación de trabajo, que se descargarán en el cliente
Html	Contiene los scripts en php que componen el sitio web del proyecto
Keys	Contiene las llaves de cifrado del proyecto
Upload	Almacena los archivos de salida de las aplicaciones enviadas por los clientes.

En el caso de las carpetas log, tmp y pid, se crearán con los nombres antes mencionados concatenados con un guion bajo seguido del nombre del servidor, en este caso, el nombre del servidor es boinc, por lo que las carpetas son nombradas log_boinc, tmp_boinc y pid_boinc.

Los directorios de subida y descarga de archivos pueden contener millones de archivos. A cada unidad de trabajo se le asigna una carpeta con un identificador único, en el cual se almacenan los archivos de entrada correspondientes dentro de la carpeta download. De la misma forma, se crea una carpeta para cada archivo de salida que llega al servidor en la carpeta upload.

Cada directorio tiene un conjunto de 1024 subdirectorios, nombrados de 0 a 3ff. Los archivos se ordenan y almacenan basándose en su nombre de archivo en estos directorios. Este método para jerarquizar las carpetas contenidas en los directorios de descarga y subida solo se emplea para organizar los archivos de entrada y salida.[57]

Existen dos formas de almacenar los archivos de entrada al directorio download, la primera es emplear el script stage_file desarrollado para colocar los archivos de

entrada en su lugar de forma manual. Para organizar los archivos de entrada de manera que los equipos voluntarios puedan acceder a ellos vía HTTP se debe correr el script stage_file de la siguiente forma:

```
bin/stage_file [--gzip] [--copy] ruta_del_archivo
```

El script mueve el archivo de entrada al directorio de descarga; este script se puede ejecutar con las siguientes opciones:
--gzip: Envía el archivo en forma comprimida a clientes 7.0+, para poder utilizar esta opción se debe incluir el atributo <gzip/> en el archivo xml de entrada de jobs.
--copy: copia el archivo de entrada al directorio de descarga en lugar de moverlo.
Este script verifica si ya existe un archivo con ese nombre, y de ser así verifica si tiene el mismo contenido a través del MD5 de dicho archivo. El script se encarga de calcular y guardar el archivo hash MD5 correspondiente al archivo de entrada.

Para conocer la ruta en la que se encuentra un archivo de entrada se ejecuta el siguiente comando:

```
bin/dir_hier_path filename
```

La segunda forma de almacenar los archivos de entrada en el directorio download, es a través del generador de trabajo, ya sea que se creen dichos archivos desde el generador o que los archivos sean copiados desde una carpeta especifica creada por el usuario al directorio download.[58]

3.2.3 ARCHIVO DE CONFIGURACIÓN CONFIG.XML

La configuración de un proyecto se describe a través de un archivo llamado config.xml contenido en el directorio de proyecto. Este archivo se crea automáticamente al crear el proyecto con el script make_project y contiene valores por defecto; a lo largo del tiempo de vida del proyecto será necesario cambiar o añadir algunos de estos valores.

El formato del archivo de configuración es el siguiente:

```
<boinc>
      <config>
            [ opciones de configuración ]
      </config>
```

```
      <daemons>
            [ lista de demonios ]
      </daemons>
      <tasks>
            [ lista de tareas periodicas]
      </tasks>
</boinc>[59]
```

3.2.4 ARCHIVO DE CONFIGURACIÓN PROJECT.XML

El archivo project.xml se describe las diferentes plataformas para las cuales se ha compilado una versión de la aplicación de trabajo. Este archivo es utilizado por el script xadd para agregar las versiones de las aplicaciones de trabajo en la base de datos de proyecto.

El formato del archivo de project.xml es el siguiente:

```
<boinc>
    <platform>
        <name>nombre_de_la_plataforma</name>
        <user_friendly_name>
           Nombre descriptivo de la plataforma
   </user_friendly_name>
    </platform>
    <app>
        <name>nombre_de_la aplicación_de_trabajo</name>
        <user_friendly_name>
               Nombre descriptivo de la aplicación de trabajo
   </user_friendly_name>
    </app>
</boinc>
```

3.2.5 APLICACIONES DE TRABAJO.

En BOINC, cuando se habla de una aplicación, no se hace referencia a un solo programa, sino a un conjunto de programas que ejecutan el mismo algoritmo pero son diferentes debido a que cada uno está dirigido a una plataforma en particular, es decir, una combinación de procesador y sistema operativo. Cada uno de estos programas se denomina versión de aplicación. Cuando se envía un trabajo, BOINC decide que versión de aplicación utilizará para realizar la computación de acuerdo a las características del cliente, estas aplicaciones reciben la información que procesarán a través de los archivos de entrada, y generan archivos de salida que el cliente enviará al servidor.[60]

Existen dos formas de portar una aplicación a BOINC, la primera es desarrollar la aplicación empleando el API de BOINC, permitiendo la manipulación de archivos y la comunicación del cliente con la aplicación, a este tipo de aplicación se le conoce como aplicación nativa.

La segunda forma es emplear el programa Wrapper que se ejecuta junto con una aplicación de la que ya se tiene un ejecutable, que puede descargarse desde la página oficial de BOINC o puede compilarse a partir del código fuente contenido en el repositorio de BOINC. La aplicación Wrapper se encarga de la comunicación con el cliente y del manejo de los archivos de entrada y de salida, de tal forma que es posible portar casi cualquier aplicación escrita en cualquier lenguaje de programación. Este método es útil cuando no se dispone del código fuente de una aplicación que se desea portar a BOINC.[61][62]

Las aplicaciones deben tener las siguientes propiedades:

- Nombre: Un nombre corto, que se utiliza también para nombrar algunos directorios del proyecto.

- Nombre amigable para el usuario: El nombre con el que los voluntarios identificarán la aplicación.

Una versión de aplicación puede constar de varios archivos, por ejemplo, un script de control, programas de pre-procesamiento y post-procesamiento y un programa principal. Debe tener un número de versión, dicho número es un entero y se debe asignar de forma ascendente, ya que BOINC utilizará únicamente la versión con el número de versión más grande.[60]

3.2.6 PROGRAMAS DAEMON DEL LADO DEL SERVIDOR

Para cada proyecto es necesario proveer al menos tres programas DAEMON que se encargarán de la manipulación de las unidades de trabajo del manejo de sus resultados, estos programas son un generador de trabajo, un validador y un asimilador.[63]

3.2.6.1 GENERADOR DE TRABAJO

El generador de trabajo se encarga de crear las unidades de trabajo y de ser necesario, de acuerdo a los requerimientos del proyecto, se encarga del manejo o creación de los archivos de entrada.
Un trabajo (job) consta de dos partes:

- Una unidad de trabajo (workunit) que define el cómputo a realizar.

- Uno o más resultados que describen una instancia de cómputo, ya sea no iniciada, en progreso o completada. Para el cliente de BOINC, estos resultados son las tareas (tasks).[64]

3.2.6.2 VALIDADOR

Cuando se obtienen los archivos de salida enviados por los clientes es necesario verificar que los resultados que contienen sean correctos, de esto se encarga el validador. Se requiere un validador por cada aplicación del proyecto. La validación se hace en dos pasos:

- Revisión de sintaxis: Verifica que los archivos de salida estén en el servidor y en el formato correcto.

- Verificación de réplicas: Si el trabajo se replica, comparar sus réplicas. Si la estricta mayoría de las réplicas es equivalente, se marcan como válidas y el resto como inválidas.

El validador se encarga de asignar la cantidad de créditos correspondiente por los trabajos realizados, de acuerdo a la cantidad de FLOPS que el cliente ha donado para procesar la unidad de trabajo.

Es posible desarrollar un validador a medida, en el que se defina una sintaxis propia para los archivos de salida o una noción propia de equivalencia, por ejemplo, definir equivalencia entre números de punto flotante. También es posible emplear uno de los tres validadores que incluye BOINC por defecto:

Tabla 3.3 Validadores incluidos en el software de BOINC

Validadores incluidos en el software	
sample_trivial_validator	Marca un trabajo como válido si sus archivos de salida estan presentes.
sample_substr_validator	Este validador lee una cadena de caracteres del archivo stderr y de acuerdo a esa cadena decide si el resultado enviado es válido o no, los casos que pueden presentarse son: • Si el archivo de salida stderr contiene una cadena especificada por el argumento de línea de comandos --stderr_string, marcará el resultado como valido. • Si el archivo de salida stderr contiene una cadena especificada por el argumento de línea de comandos --reject_if_present marcará el resultado como invalido.
sample_bitwise_validator	Los archivos de salida son válidos si concuerdan byte por byte. Este validador se puede utilizar si la aplicación genera resultados exactos, por ejemplo, cuando no se realizan cálculos con punto flotante, o por que se utiliza redundancia homogénea.

Si los archivos de salida estan comprimidos (gzip) se debe utilizar el argumento de línea de comandos --is_gzip, para saltar la cabecera gzip al realizar la comparación.[65]

3.2.6.3 ASIMILADOR

El asimilador se encarga del manejo de los trabajos completados, ya sea que solo se requiera almacenarlos en una base de datos o se requiera realizar alguna otra

acción con los resultados obtenidos. Se debe desarrollar uno específico para cada aplicación.[66]

3.2.7 CREACIÓN DE UN PROYECTO NUEVO VACÍO

El software de BOINC contiene un script que se emplea para crear nuevos proyectos, este script crea los componentes básicos del proyecto.

En el servidor BOINC, en una terminal se ejecutan los siguientes comandos como usuario normal (sin privilegios de administrador).

3.2.7.1 SCRIPT MAKE_PROJECT

Los pasos a seguir para crear un nuevo proyecto vacío se mencionan a continuación:

Entrar a la carpeta tools del software de BOINC:

```
cd boinc_src/tools
```

Ejecutar el script de creación de proyectos:

```
./make_project appliz1
```

En la figura 3.19 se muestra la captura de pantalla de la creación del proyecto con el comando make project.

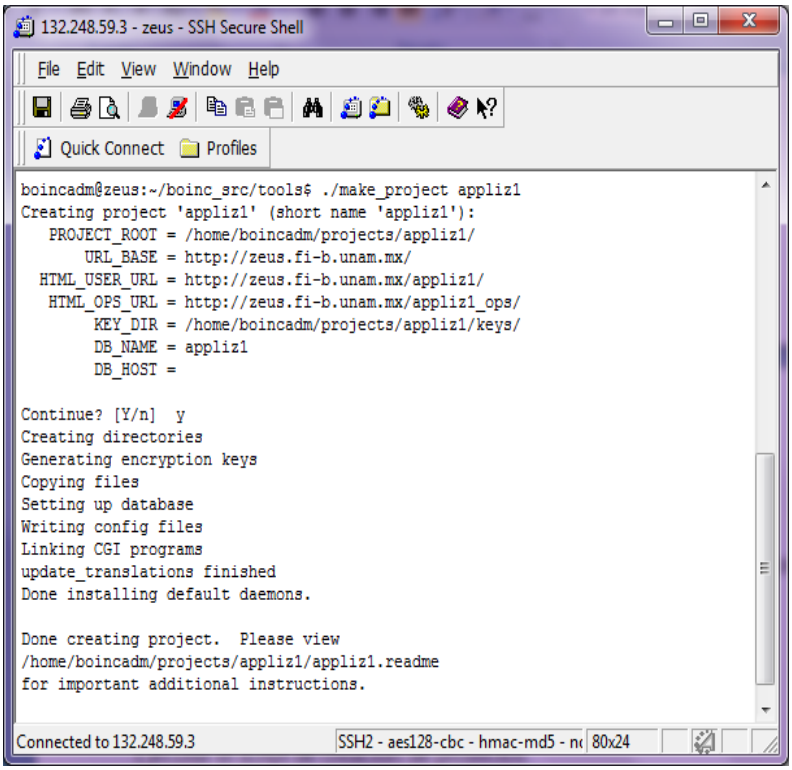


Figura 3.19 Ejecución del script make_project

Se crea un proyecto vacío, con los archivos y demonios por defecto, se copian a la carpeta bin todos los validadores, el asimilador por defecto y el generador de trabajo por defecto.

3.2.7.2 CREACIÓN DE DIRECTORIOS PARA LAS DISTINTAS VERSIONES DE LA APLICACIÓN DE TRABAJO

El siguiente paso es copiar los ejecutables de las diferentes versiones de las aplicaciones de trabajo a la carpeta apps, en este caso, la versión para procesadores intel con S.O. Fedora de 64 bits y la versión para sparc de 64 bits con Debian. Se debe crear la siguiente estructura de directorios manualmente como se muestra en la figura 3.20:

```
apps/
     appliz1/
          1.0/
               x86_64-pc-linux-gnu/
               sparc64-sun-linux-gnu/
```

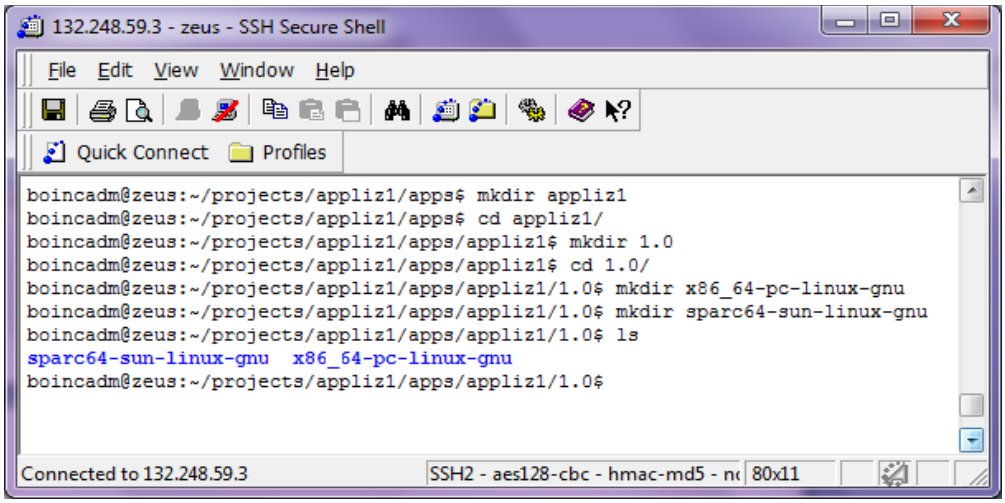


Figura 3.20 Creación de los directorios

Las carpetas donde se guardarán los ejecutables deben nombrarse como la plataforma a la que va dirigida cada versión. La forma de nombrar las plataformas hacia las que va dirigida cada versión de la aplicación de trabajo.

3.2.7.3 COPIA DE LAS APLICACIONES DE TRABAJO A SUS RESPECTIVOS DIRECTORIOS

El siguiente paso es copiar las versiones de las aplicaciones a las carpetas correspondientes, los ejecutables deben ser nombrados de la siguiente manera:

```
nom-app_#version_plataforma
```

Para la versión para sparc de 64 bits con Debian el nombre es el siguiente:

```
appliz1_1.0_sparc64-sun-linux-gnu
```

En la figura 3.21, puede observarse la aplicación renombrada para la plataforma sparc de 64 bits con sistema operativo GNU/Linux.[67]

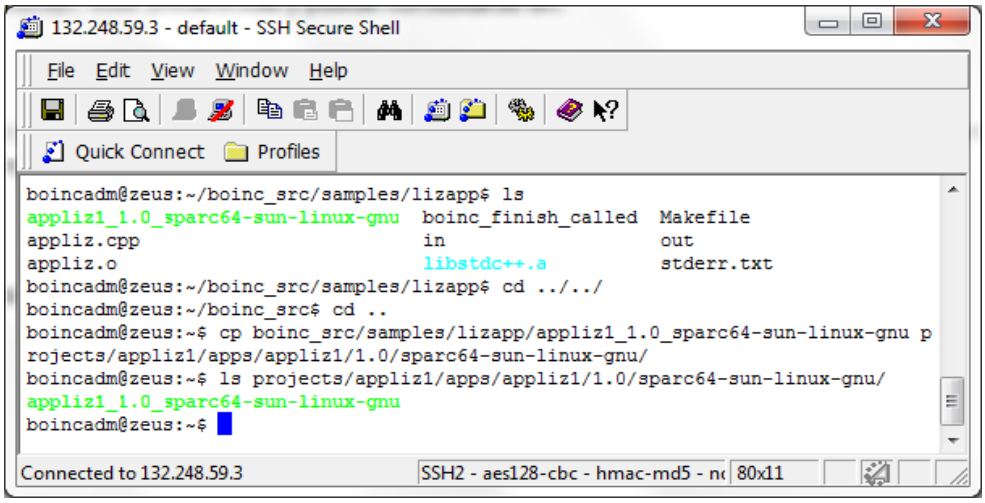


Figura 3.21 Copia del ejecutable para sparc64

3.2.7.4 CREACIÓN DE ARCHIVOS DE DESCRIPCIÓN DE CADA VERSIÓN

También se requiere de un archivo de descripción de la versión de la aplicación, que se almacena en el mismo directorio que la aplicación.

El contenido del archivo version.xml para la plataforma sparc de 64 bits con sistema operativo GNU/Linux es [67]:

```
<version>
      <file>
            <physical_name>
                  appliz1_1.0_sparc64-sun-linux-gnu
            </physical_name>
            <main_program/>
            <logical_name>appliz1</logical_name>
      </file>
</version>
```

3.2.7.5 CONFIGURACIÓN DE LAS VERSIONES EN EL ARCHIVO PROJECT.XML

Una vez agregadas las aplicaciones con sus respectivos archivos de descripción de versión, se debe configurar el archivo project.xml que se encuentra en la carpeta principal del proyecto [68].

El contenido del archivo project.xml para este proyecto es:

```
<boinc>
    <platform>
      <name>x86_64-pc-linux-gnu</name>
      <user_friendly_name>
            Linux running on an AMD x86_64 or Intel EM64T  CPU
      </user_friendly_name>
    </platform>
    <platform>
      <name>sparc64-sun-linux-gnu</name>
      <user_friendly_name>
            Linux running on a SPARC 64-bit CPU
      </user_friendly_name>
    </platform>
    <app>
      <name>appliz1</name>
      <user_friendly_name>Aplicacion de Liz</user_friendly_name>
    </app>
</boinc>
```

3.2.7.6 CONFIGURAR EL PROYECTO EN EL SERVIDOR APACHE

El siguiente paso es añadir el proyecto al servidor apache, para esto se puede copiar el contenido del archivo httpd.conf creado para él proyecto o copiando la ruta completa de dicho archivo al archivo etc/apache2/apache2.httpd.conf, los archivos de conexión para apache se crean con un nombre de este tipo:

```
nom_proyecto.httpd.conf
```

En este caso el archivo se llama appliz1.httpd.conf para este paso se requiere tener permisos de administrador (root), la conexión del proyecto en apache se muestra en la figura 3.22.

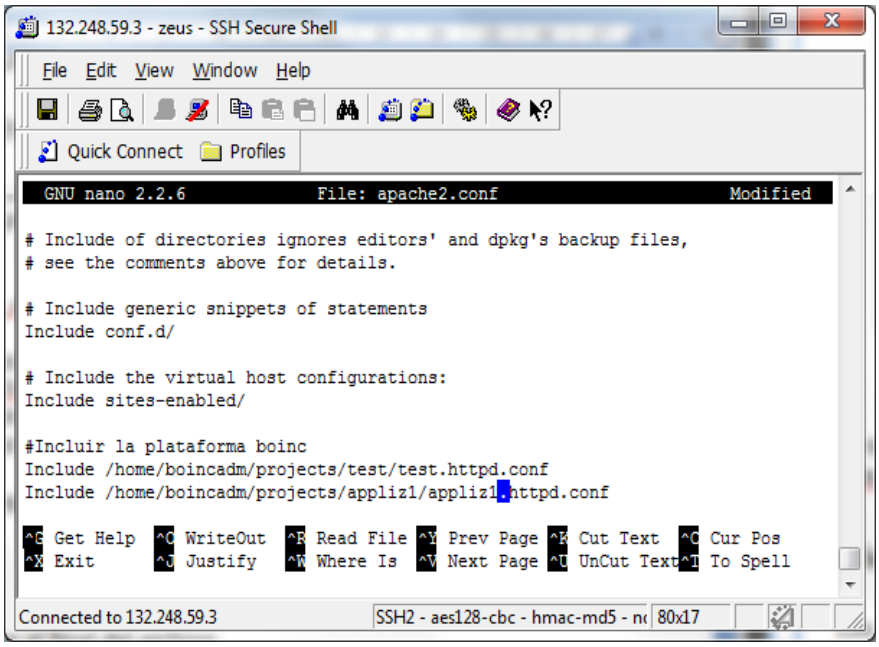


Figura 3.22 Añadir el proyecto al servidor Apache

Después se debe reiniciar el servidor apache como se ilustra en la figura 3.23 con el comando (en Debian):

```
service apache2 restart
```

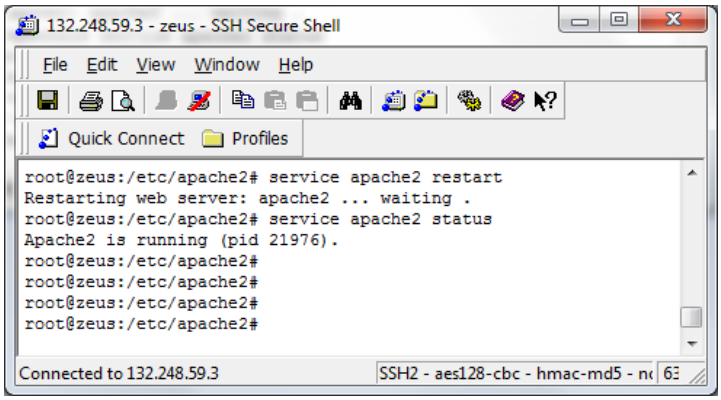


Figura 3.23 Reinicio del servidor Apache

3.2.7.7 AÑADIR EL PROYECTO A CRONTAB PARA SU EJECUCIÓN PERIÓDICA.

Como usuario normal se añade el proyecto al archivo crontab, como se muestra en la figura 3.24, para que se ejecute periódicamente, se emplean los siguientes comandos:

```
crontab -e
```

Se debe añadir la siguiente línea al archivo:

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * cd
/home/boincadm/projects/appliz1 ;
/home/boincadm/projects/appliz1/bin/start --cron
```

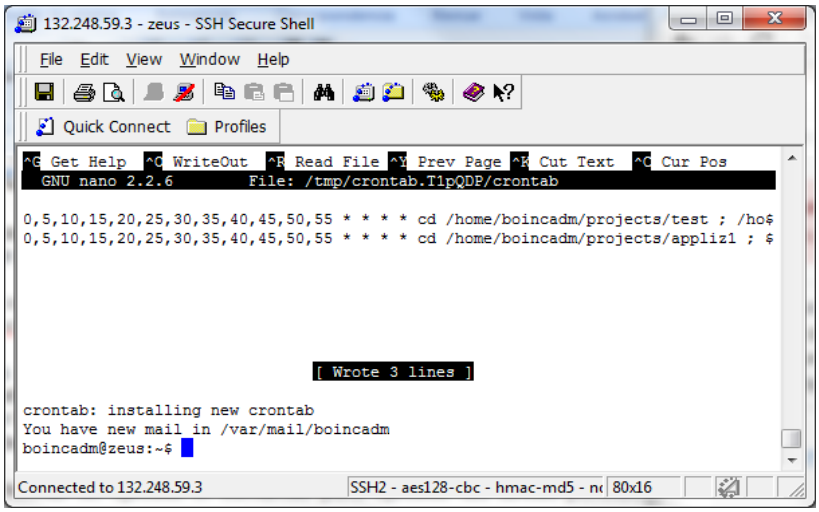


Figura 3.24 Añadir el proyecto a crontab

Después de realizar estos pasos la página de acceso al proyecto esta lista y visible desde internet, como se muestra en la figura 3.25. Esta página se crea también cuando se crea el proyecto con el script make_project

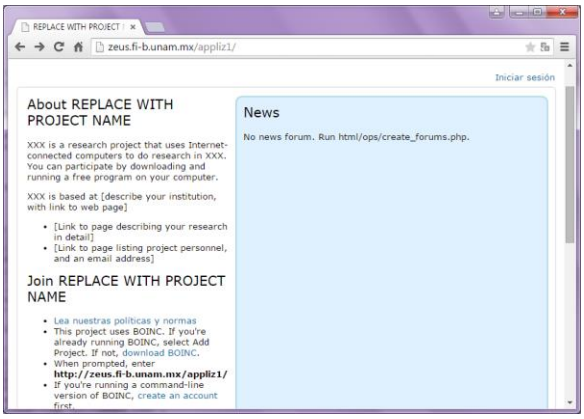


Figura 3.25 Página web creada automáticamente por el script make_project

3.2.7.8 INSTALACIÓN DE LA APLICACIÓN Y CONFIGURACIÓN EN LA BASE DE DATOS

El siguiente paso para la creación del proyecto es instalar la aplicación e inicializar la base de datos. Se ejecutan los siguientes comandos:

```
bin/xadd ; bin/update_versions
```

El comando xadd inicializa la base de datos y configura las plataformas de las versiones existentes, su ejecución se muestra en la figura 3.26.

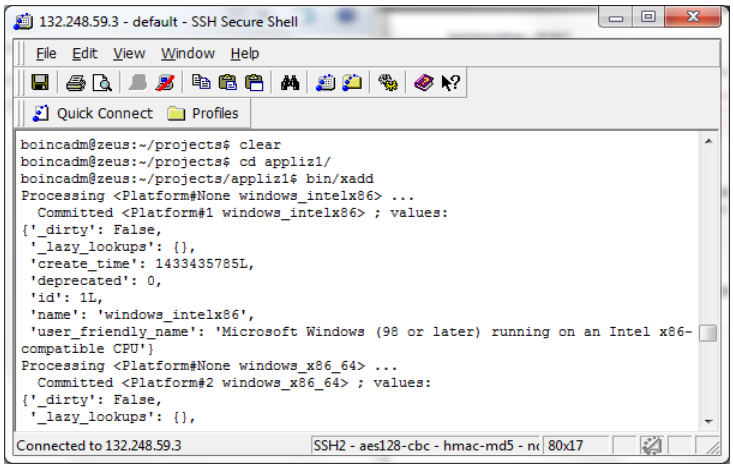


Figura 3.26 Ejecución del comando bin/xadd

El comando update_versions carga los archivos al directorio de descarga del proyecto y crea registros de las nuevas versiones de la aplicación en la base de datos, su ejecución se muestra en la figura 3.27 [54].

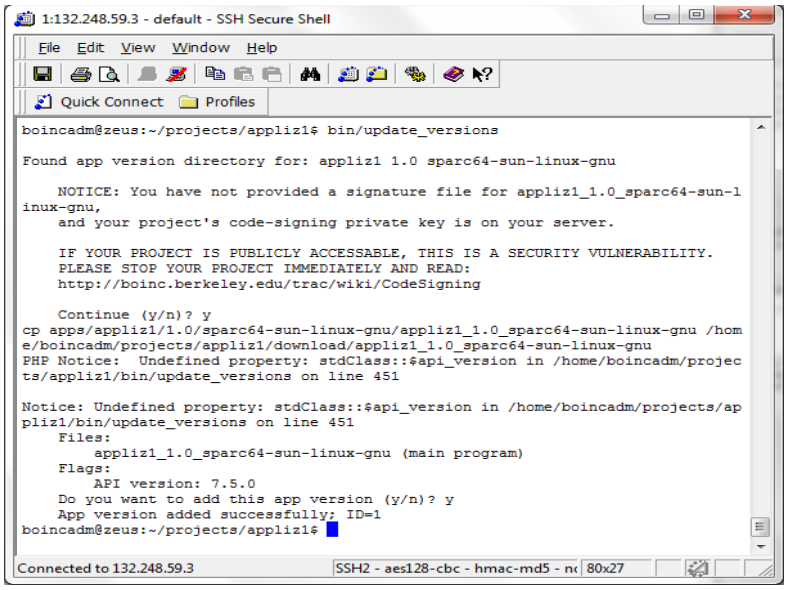


Figura 3.27 Ejecución de bin/update_versions

Se pueden añadir nuevas versiones de aplicaciones, añadiéndolas al archivo project.xml y creando sus respectivas carpetas en el directorio apps y ejecutando de nuevo los comandos xadd y update_versions, como se muestra en la figura 3.28.[68]

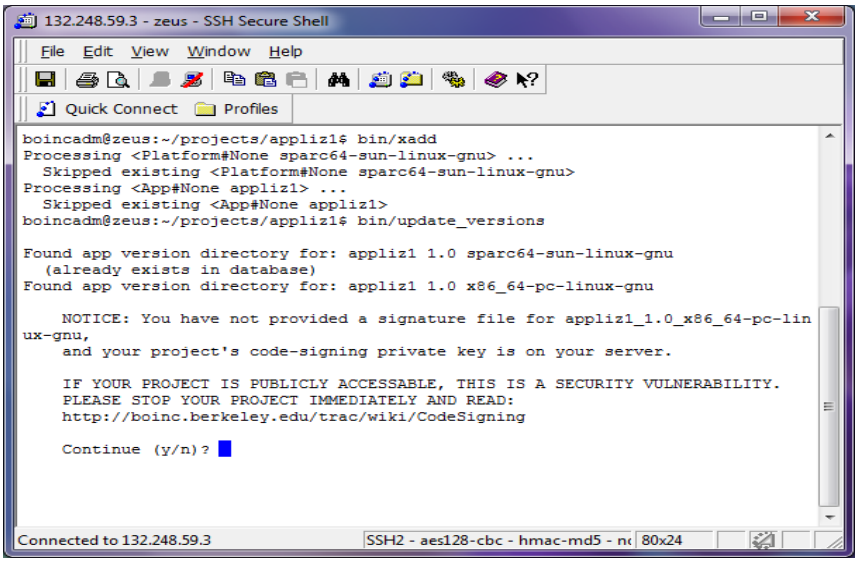


Figura 3.28 Ejecución de los comandos xadd y update_versions para agregar la versión para procesadores intel y AMD compilada en Fedora

Una vez instaladas las aplicaciones podemos comenzar la ejecución del proyecto con el comando bin/start.

3.2.7.9 AÑADIR LOS ARCHIVOS DE ENTRADA

Los archivos de entrada se suben utilizando el script stage_file, como se muestra en la figura 3.29.

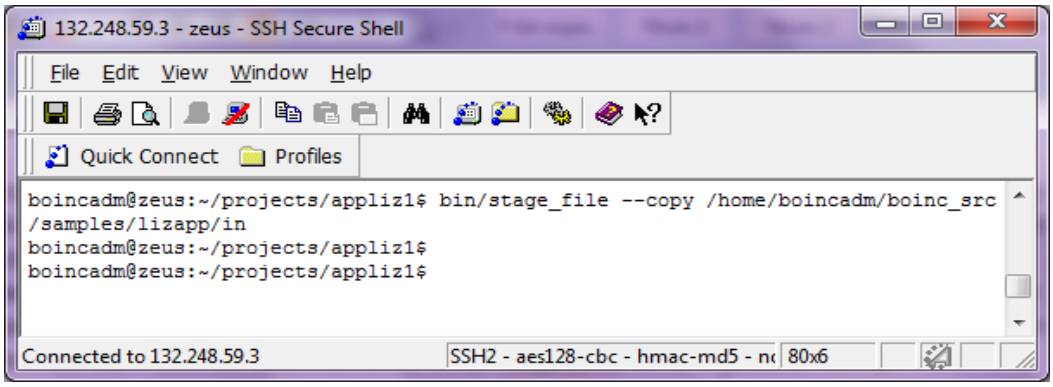


Figura 3.29 Copia del archivo de entrada con el script stage_file

3.2.7.10 CREACIÓN DE LOS TEMPLATES DE LOS ARCHIVOS DE ENTRADA Y SALIDA

Una vez guardados los archivos de entrada en el servidor, se deben crear los templates xml de entrada y salida para los trabajos, y se deben guardar en el directorio templates.

El contenido del template de entrada es el siguiente:

```
<input_template>
      <file_info>
            <number>0</number>
      </file_info>
      <workunit>
            <file_ref>
                  <file_number>0</file_number>
                  <open_name>in</open_name>
            </file_ref>
            <command_line>-cpu_time 30</command_line>
      </workunit>
</input_template>
```

El contenido del template de salida es el siguiente:

```
<output_template>
      <file_info>
```

```
            <name><OUTFILE_0/></name>
            <generated_locally/>
            <upload_when_present/>
            <max_nbytes>5000000</max_nbytes>
            <url><UPLOAD_URL/></url>
      </file_info>
      <result>
            <file_ref>
                  <file_name><OUTFILE_0/></file_name>
                  <open_name>out</open_name>
            </file_ref>
      </result>
</output_template>[69]
```

CAPITULO 4: DESARROLLO DE APLICACIONES EN SISTEMAS GNU/LINUX.

En BOINC una plataforma es un objetivo de compilación para una aplicación, usualmente el termino plataforma se refiere a una combinación de una arquitectura de CPU y un Sistema Operativo.

Las arquitecturas elegidas como entornos de desarrollo para sistemas con GNU/Linux son las siguientes:

- ✓ Procesadores Intel de 32 bits
- ✓ Procesadores Intel de 64 bits
- ✓ Procesadores PowerPC de 64 bits
- ✓ Procesadores Sparc de 64 bits

4.1 CONFIGURACION DE SISTEMAS CON PROCESADORES INTEL DE 32 Y 64 BITS.

Se utilizó el sistema operativo Fedora 20 para el desarrollo de aplicaciones para BOINC en sistemas GNU/Linux con procesadores Intel, todos los paquetes de las dependencias a resolver se instalaron por consola empleando el gestor de paquetes “yum”, los nombres de los paquetes son los mismos tanto para la versión de 32 bits como para la de 64 bits, yum instala la versión correcta para cada arquitectura.

4.1.1 CONFIGURACIÓN DEL ENTORNO DE DESARROLLO.

Para comenzar con el desarrollo de aplicaciones para BOINC se deben cumplir tres requisitos:

1. Resolver las dependencias para configurar el entorno.
2. Descargar el software de BOINC
3. Compilar y configurar el software de BOINC

Estos requisitos son los mismos para todas las plataformas, no solo para arquitecturas Intel.

4.1.2 DEPENDENCIAS

Dado que el entorno solo será para el desarrollo de aplicaciones, solo se requieren los siguientes prerrequisitos:

- git – Para obtener el software de BOINC.
- Herramientas de GNU: make, m4, libtool, autoconf, automake, gcc, gcc-c++.
- pkgconfig
- freeglut, freeglut-devel – Para el desarrollo de aplicaciones gráficas. Dado que el paquete compilado no contiene las librerías estáticas (con extensión .a), se debe compilar freeglut desde código fuente.
- libjpeg-turbo-devel, xorg-x11-server-devel, libXmu-devel, libXi-devel, ftgl-devel – Para el desarrollo de aplicaciones gráficas.
- libstdc++.a (libstdc++-static) – Para crear aplicaciones portables.

4.1.3 DESCARGA DEL SOFTWARE DE BOINC

El software se descarga a través de git con el comando, la figura 4.1 muestra la descarga del software:

```
git clone git://boinc.berkeley.edu/boinc-v2 boinc
```

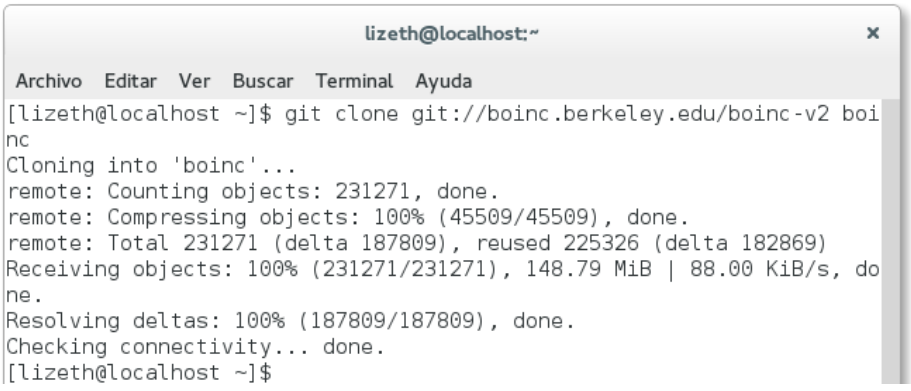


Figura 4.1 Obtención de software de BOINC con Git desde terminal de Fedora 20

4.1.4 COMPILAR Y CONFIGURAR EL SOFTWARE DE BOINC

Dado que el entorno solo es de desarrollo se compila y configura con los siguientes comandos:

```
./_autosetup
./configure --disable-server --disable-client \
      --disable-manager
make
```

El comando ./_autosetup, cuya salida se muestra en la figura 4.2 genera el script configure.

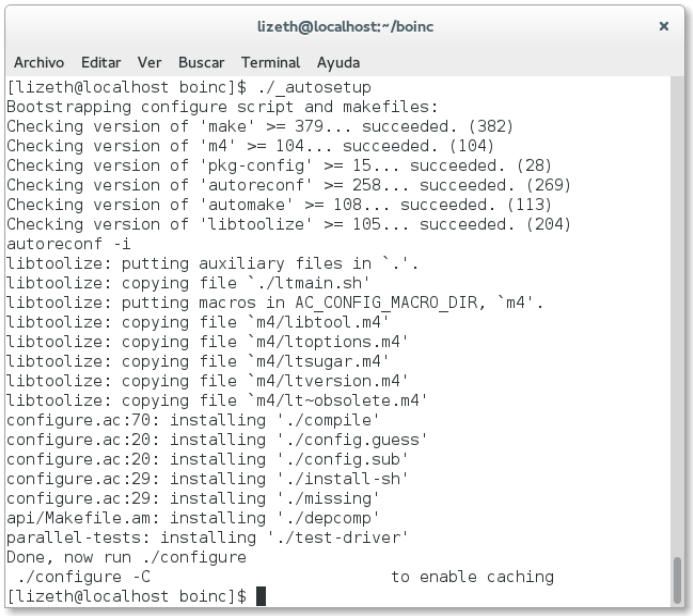


Figura 4.2 Configuración del entorno de desarrollo en Fedora 20 – script autosetup

El comando ./configure con la configuración antes mencionada creará los makefiles para compilar solamente el API de BOINC, ya que se ha deshabilitado la compilación del software del servidor, del cliente y del administrador, la salida del comando configure se muestra en la figura 4.3.

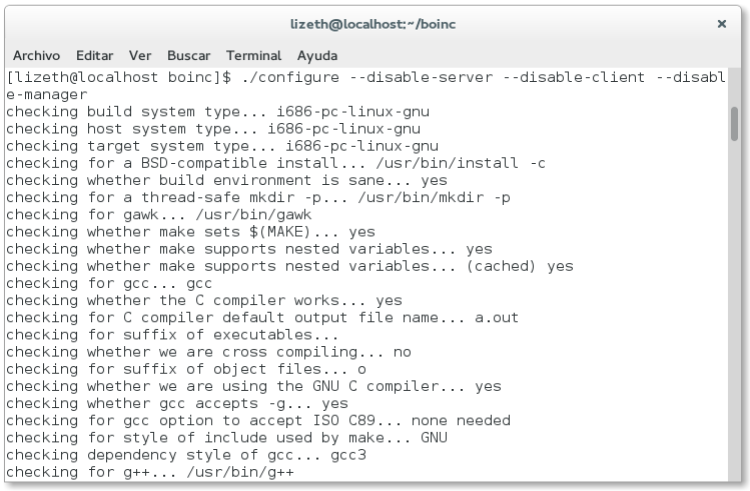


Figura 4.3 Configuración del entorno de desarrollo en Fedora 20 – script configure

Finalmente se ejecuta el comando make para compilar el API, la salida del comando make se muestra en la figura 4.4.

```
[lizeth@localhost boinc]$ make
cd . && sh generate_svn_version.sh
make  all-recursive
make[1]: se ingresa al directorio `/home/lizeth/boinc'
Making all in m4
make[2]: se ingresa al directorio `/home/lizeth/boinc/m4'
make[2]: No se hace nada para `all'.
make[2]: se sale del directorio `/home/lizeth/boinc/m4'
Making all in api
make[2]: se ingresa al directorio `/home/lizeth/boinc/api'
  CXX      boinc_api.lo
  CXX      graphics2_util.lo
  CXX      reduce_main.lo
  CXXLD    libboinc_api.la
  CXX      boinc_opencl.lo
  CXXLD    libboinc_opencl.la
```

Figura 4.4 Configuración del entorno de desarrollo en Fedora 20 - make

4.1.5 COMPROBACIÓN DEL ENTORNO (COMPILACIÓN DE LA APLICACIÓN DE PRUEBA)

Para comprobar si el entorno se ha configurado satisfactoriamente se puede compilar la aplicación de ejemplo contenida en el directorio boinc/samples/example_app

Lo primero es hacer un link a la librería libstdc++.a con el comando:

```
ln -s `g++ -print-file-name=libstdc++.a`
```

Después se ejecuta el comando make para compilar el código fuente con el makefile, en la figura 4.5 se muestra la compilación de la aplicación de prueba.

Figura 4.5 Compilación de la aplicación de prueba

4.1.6 EJECUCIÓN DE LA APLICACIÓN DE PRUEBA

La prueba de la aplicación de ejemplo cambia a mayúsculas el contenido del archivo de entrada y requiere de la creación de un archivo de entrada, el archivo con el que se realizó la prueba se muestra en la figura 4.6 junto con el contenido de la carpeta example_app, donde se puede ver los ejecutables generados en la compilación en color verde.

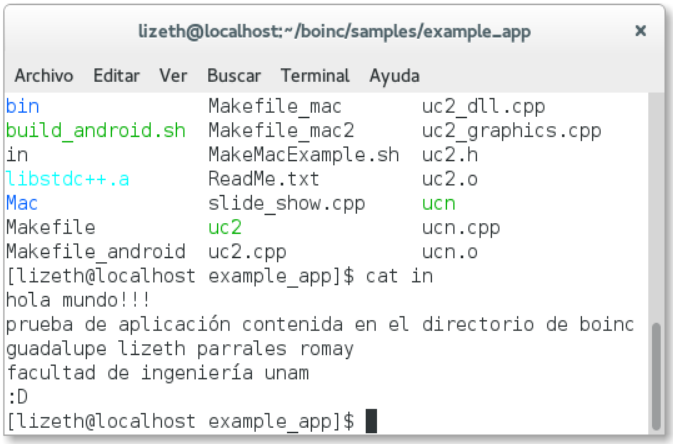


Figura 4.6 Contenido de la carpera example_app y del archivo de prueba in

Se crea el archivo out (también es el nombre lógico), y se crea el archivo stderr.txt, donde se guardan mensajes de la ejecución de la aplicación, en la figura 4.7 se muestra de nuevo el contenido de la carpeta example_app, en donde se ha creado el archivo out, además se muestra el contenido del archivo out.[70]

Figura 4.7 Ejecución de la aplicación de prueba uc2 y contenido del archivo de salida out

4.2 CONFIGURACIÓN DE SISTEMAS DE PROCESADORES POWERPC DE 64 BITS

Para el desarrollo de aplicaciones en la arquitectura PowerPC de 64 bits se emplearon sistemas PlayStation3 y XBOX 360. Aunque ambos son procesadores PowerPC de 64 bits, la arquitectura interna de sus procesadores es completamente diferente. El procesador del PlayStation3 es un procesador CELL, que está formado por un procesador PowerPC y que además cuenta con 8 Elementos de Procesamiento Sinérgicos (SPE). Por otro lado, el procesador del XBOX 360 está formado por tres núcleos PowerPC de 3.2 GHz que comparten un módulo de memoria cache de 1 MB.

4.2.1 PLAYSTATION 3

El PlayStation 3 (PS3) cuenta con un procesador PowerPC de 64 bits; para esta plataforma se utilizó una distribución de GNU/Linux llamada Red Ribbon que está basada en Debian, la versión utilizada está basada en Debian 8 (Jessie).

4.2.1.1 CONFIGURACIÓN DEL ENTORNO DE DESARROLLO.

La descarga e instalación de los paquetes necesarios para resolver las dependencias de BOINC se hace por medio del administrador de paquetes apt (Advance Packaging Tool). Debemos asegurarnos de que el archivo /etc/apt/source.list contenga las direcciones de todos los repositorios de debían y red ribbon para que el administrador de paquetes apt sea capaz de encontrar todos los paquetes a instalar, el contenido del archivo source.list con la lista completa de los repositorios se muestra en la figura 4.8.

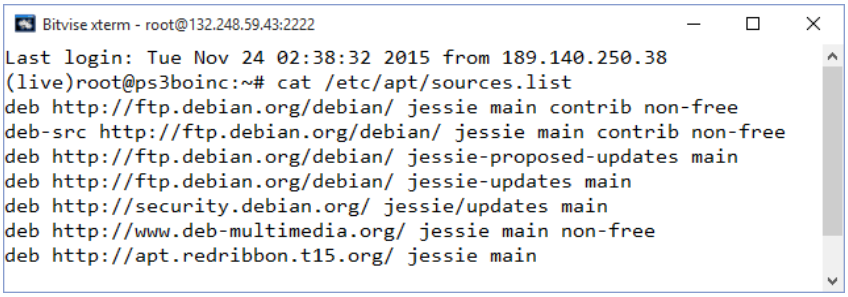


Figura 4.8 Contenido del archivo source.list

Una vez modificado el archivo de repositorios, se descarga la lista de paquetes disponibles en los repositorios ejecutando el comando:

```
apt-get update
```

El siguiente paso es descargar las dependencias para la configuración del entorno de desarrollo.

4.2.1.2 DEPENDENCIAS

- ✓ git – Para obtener el software de BOINC.
- ✓ Herramientas de GNU: make, gcc, g++, autoconf, automake, libtool, m4.
- ✓ pkg-config
- ✓ libstdc++.a (libstdc++-x.x-dev) – Para crear aplicaciones portables. En este caso se instaló la versión 4.9 (libstdc++-4.9-dev) la versión de esta librería debe ser la misma que la del paquete g++.

4.2.1.3 DESCARGA DEL SOFTWARE DE BOINC

El siguiente paso es la descarga del software de BOINC, debido a que esta configuración es más reciente que la de los sistemas con procesadores Intel, el nombre del repositorio cambió, actualmente la descarga de BOINC se hace a través del siguiente comando:

```
$ git clone https://github.com/BOINC/boinc boinc
```

La descarga del software se muestra en la figura 4.9.

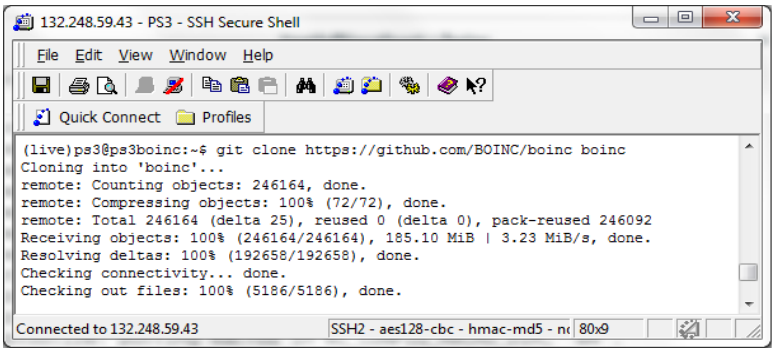


Figura 4.9 Descarga del software de BOINC

4.2.1.4 COMPILAR Y CONFIGURAR EL SOFTWARE DE BOINC

Se ejecuta el comando ./_autosetup que verifica que se cumpla con las dependencias mínimas necesarias, además de verificar que las versiones de los paquetes sean las correctas, esto se ilustra en la figura 4.10.

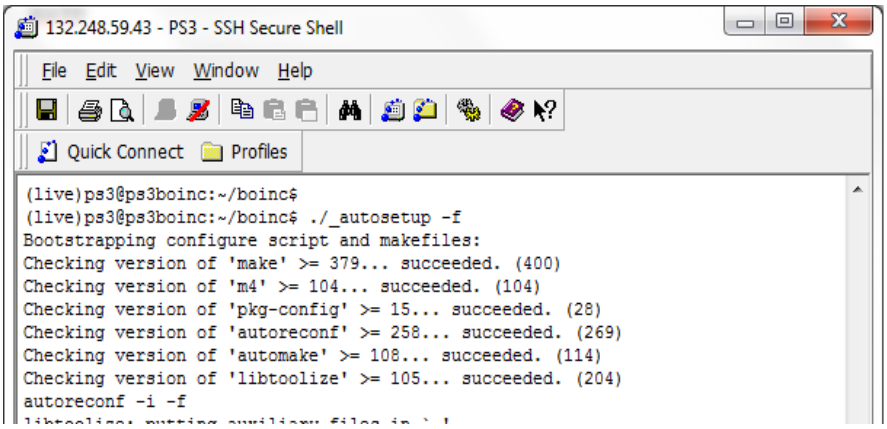


Figura 4.10 Salida del script _autosetup

El comando ./_autosetup genera el script ./configure, debido a que solo se configuró un entorno de desarrollo se deshabilitan los componentes que no son necesarios, es decir, el software del servidor, el cliente y el administrador, de esta forma solo se compilará el API. El comando es el siguiente, para obtener las librerías portables, (con estensión .a), se debe copilar el código fuente de freeglut en lugar de descargar el paquete de los repositorios, esto sucede tanto en debian como en fedora, si se emplea el paquete, el script configure mandará una advertencia como la que se muestra en la figura 4.11.

```
$ ./configure --disable-server --disable-client \
--disable-manager
```

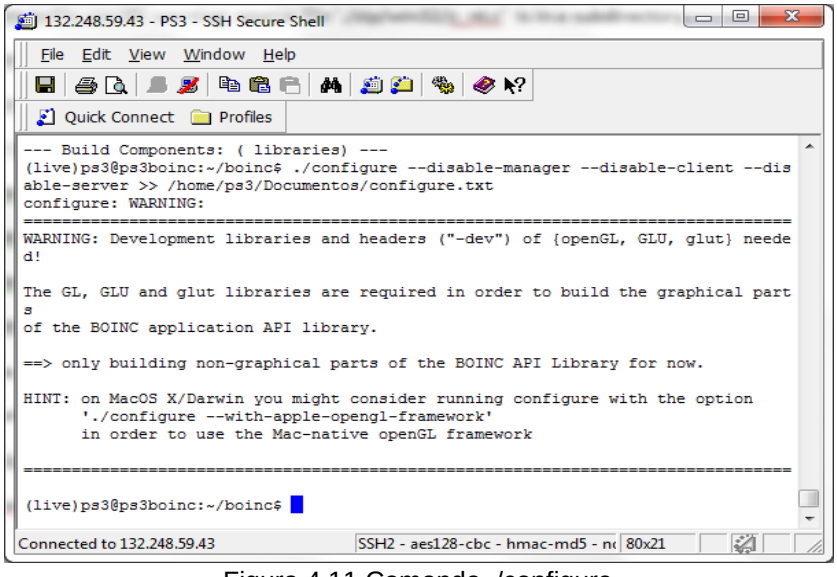


Figura 4.11 Comando ./configure

Una vez configurado el entorno se ejecuta el comando make para compilar el API de BOINC como se muestra en la figura 4.12.

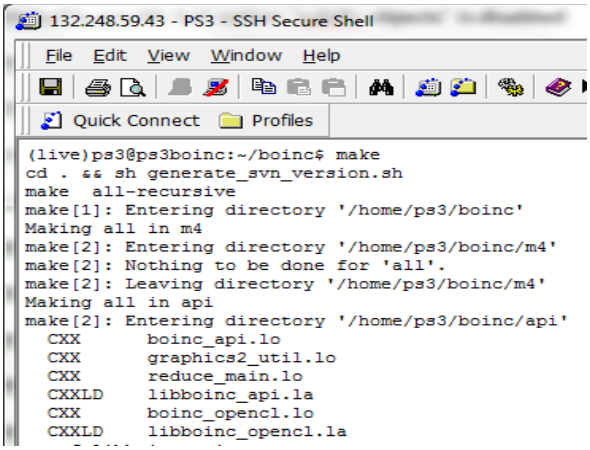


Figura 4.12 Salida del comando make

4.2.1.5 COMPROBACIÓN DEL ENTORNO

Se comprueba si el entorno se ha configurado satisfactoriamente compilando la aplicación de ejemplo contenida en el directorio boinc/samples/example_app, se crea un enlace simbólico a la librería libstdc++.a con el comando:

```
ln -s `g++ -print-file-name=libstdc++.a`
```

Una vez creado el enlace simbólico, se ejecuta el comando make para compilar el código fuente con el makefile, como se muestra en la figura 4.13.

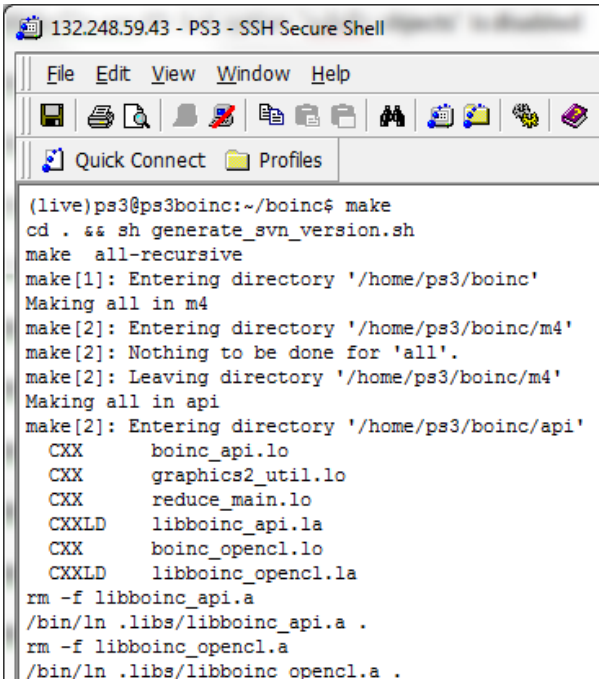


Figura 4.13 Compilación de la aplicación de ejemplo en el PS3

4.2.2 XBOX 360

El XBOX 360 utiliza un procesador PowerPC de 3 nucleos de 3.2 GHz cada uno (conocido por el nombre Xenon). El sistema utiliza el sistema operativo Ubuntu 7.10 (Gutsy Gibbon), empleando XeLL (Xenon Linux Loader) para la carga del mismo.

4.2.2.1 CONFIGURACIÓN DEL ENTORNO DE DESARROLLO.

Esta distribución emplea el administrador de paquetes apt, por lo tanto es necesario verificar que el archivo /etc/apt/source.list contenga el listado completo de los repositorios disponibles para esta versión del sistema operativo, el contenido del archivo source.list se muestra a continuación:

```
deb http://old-releases.ubuntu.com/ubuntu/ gutsy main restricted
deb-src http://old-releases.ubuntu.com/ubuntu/ gutsy main restricted
deb http://old-releases.ubuntu.com/ubuntu/ gutsy-updates main restricted
deb-src http://old-releases.ubuntu.com/ubuntu/ gutsy-updates main
restricted
deb http://old-releases.ubuntu.com/ubuntu/ gutsy universe
deb-src http://old-releases.ubuntu.com/ubuntu/ gutsy universe
deb http://old-releases.ubuntu.com/ubuntu/ gutsy-updates universe
deb-src http://old-releases.ubuntu.com/ubuntu/ gutsy-updates universe
deb http://old-releases.ubuntu.com/ubuntu/ gutsy multiverse
deb-src http://old-releases.ubuntu.com/ubuntu/ gutsy multiverse
deb http://old-releases.ubuntu.com/ubuntu/ gutsy-updates multiverse
deb-src http://old-releases.ubuntu.com/ubuntu/ gutsy-updates multiverse
 deb http://old-releases.ubuntu.com/ubuntu/ gutsy-backports main
restricted universe multiverse
 deb-src http://old-releases.ubuntu.com/ubuntu/ gutsy-backports main
restricted universe multiverse
 deb http://archive.canonical.com/ubuntu gutsy partner
 deb-src http://archive.canonical.com/ubuntu gutsy partner
deb http://old-releases.ubuntu.com/ubuntu gutsy-security main restricted
deb-src http://old-releases.ubuntu.com/ubuntu gutsy-security main
restricted
deb http://old-releases.ubuntu.com/ubuntu gutsy-security universe
deb-src http://old-releases.ubuntu.com/ubuntu gutsy-security universe
deb http://old-releases.ubuntu.com/ubuntu gutsy-security multiverse
deb-src http://old-releases.ubuntu.com/ubuntu gutsy-security multiverse
```

Para cargar la lista de paquetes se ejecuta el comando:

```
apt-get update
```

4.2.2.2 DEPENDENCIAS

- ✓ git – Para obtener el software de BOINC.
- ✓ Herramientas de GNU: make, gcc, g++ (versión 4.2), autoconf, automake, libtool, m4.
- ✓ pkg-config
- ✓ libstdc++.a – Para crear aplicaciones portables. Para la instalación de esta librería en este sistema se debe instalar la librería libstdc++6-4.2-dbg, la librería libstdc++6-4.2-dev se instala automáticamente al instalar g++.

4.2.2.3 DESCARGA DEL SOFTWARE DE BOINC

El siguiente paso es la descarga del software de BOINC con el siguiente comando:

```
git clone https://github.com/BOINC/boinc boinc
```

4.2.2.4 COMPILAR Y CONFIGURAR EL SOFTWARE DE BOINC

El entorno solo es de desarrollo, de tal forma que solo se necesita compilar el API utilizando los siguientes comandos, en la figura 4.14 se muestra parte de la salida del script configure:

```
./_autosetup
./configure --disable-server --disable-client \
      --disable-manager
make
```

```
checking build system type... powerpc64-unknown-linux-gnu
checking host system type... powerpc64-unknown-linux-gnu
checking target system type... powerpc64-unknown-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... /usr/bin/gawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for style of include used by make... GNU
```

Figura 4.14 Captura de pantalla de la salida del comando ./configure

4.2.2.5 COMPROBACIÓN DEL ENTORNO

Se comprueba si el entorno se ha configurado satisfactoriamente está vez compilando una aplicación de propia que se guardó en el directorio samples boinc/samples/appliz. Lo primero que se debe hacer es crear un enlace simbólico a la librería libstdc++.a con el comando:

```
ln -s `g++ -print-file-name=libstdc++.a`
```

Una vez creado el enlace simbólico, se ejecuta el comando make para compilar el código fuente con el makefile, lo que se ilustra en la figura 4.15.

```
make: Nothing to be done for `all'.
g++ -g -I../.. -I../../lib -I../../api -L.   -c -o appliz.o appliz.cpp
ln -s `g++ -print-file-name=libstdc++.a`
g++ -g -I../.. -I../../lib -I../../api -L.   -o appliz appliz.o
libstdc++.a -pthread \
      ../../api/libboinc_api.a \
      ../../lib/libboinc.a
```

Figura 4.15 Compilación de la aplicación appliz

4.3 DESARROLLO DE APLICACIONES DE TRABAJO

Para el desarrollo de aplicaciones de trabajo, son necesarias las siguientes librerías de la tabla 4.1.

Tabla 4.1 Librerías básicas necesarias para el desarrollo de aplicaciones [71]

Librerías básicas necesarias para el desarrollo de aplicaciones
boinc_api.h
Es la librería básica del API de BOINC, es una colección de funciones escritas en C++, que en su mayoría tienen una interfaz en lenguaje C, lo que permite que puedan ser utilizadas en aplicaciones escritas en lenguaje C o en algún otro lenguaje de programación que permita añadir código en lenguaje C. Contiene funciones de inicialización especiales para aplicaciones secuenciales (aplicaciones que se ejecutan en un solo hilo), y para aplicaciones paralelas (multihilo). El API de BOINC también provee mecanismos de control para trabajos de gran tamaño de los que es necesario hacer un seguimiento del estado de la ejecución en un periodo de tiempo determinado.
filesys.h
Es una librería que se encarga de la manipulación de archivos y directorios. Cuenta con una función especial para abrir archivos boinc_fopen(), a diferencia de la función fopen convencional del lenguaje C, boinc_open() hace varios intentos de abrir los archivos de entrada debido a que a veces los sistemas operativos bloquean los archivos temporalmente. Además provee una clase, DirScanner, que permite hacer escaneo de directorios en busca de archivos.
mfile.h
Provee una clase especial para la creación de archivos, MFILE, que guarda en un buffer el contenido que se escribirá el archivo, y lo escribe hasta que la aplicación ejecuta la instrucción de cerrar el archivo.

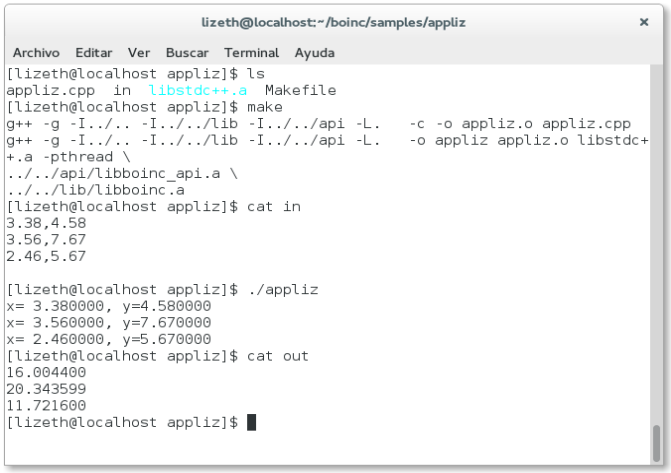
4.3.1 Compilación la aplicación de trabajo

Además del código fuente de la aplicación se necesita hacer un makefile para compilar el código fuente con todas las dependencias necesarias. El makefile para este código fuente se muestra en el anexo B.

Ambos archivos se guardan en una carpeta, en la cual hay que crear el link a la librería libstdc++.a con el comando:

```
ln -s `g++ -print-file-name=libstdc++.a`
```

Y se compila la aplicación con el comando make, como se muestra en la figura 4.16.



4.16 Compilación y ejecución de aplicación de prueba desarrollada en PC Intel de 64 bits

En la figura 4.17 se muestra la compilación de la aplicación de trabajo para la arquitectura sparc de 64 bits.

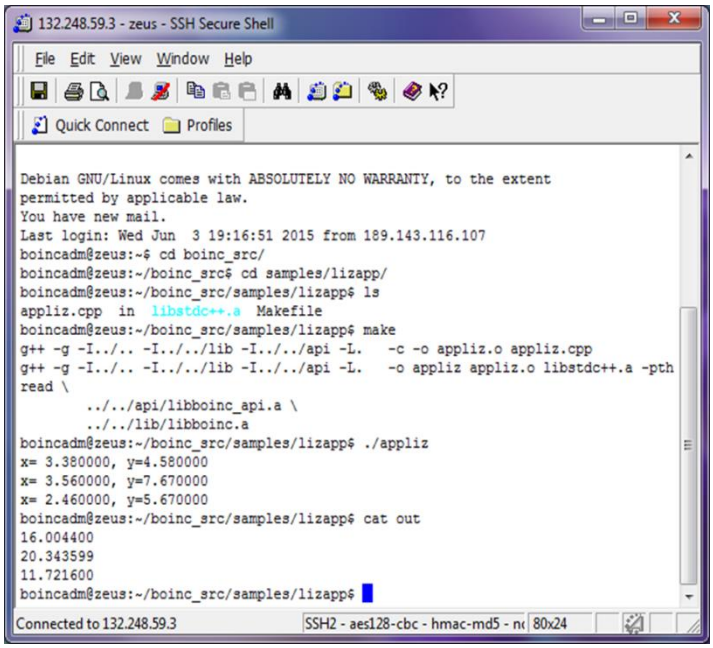


Figura 4.17 Compilación de la aplicación de trabajo de forma remota en un servidor BOINC con procesador sparc de 64 bits y S.O. GNU/Linux Debian.

Y en la figura 4.18 se muestra la compilación de la aplicación en un PS3.

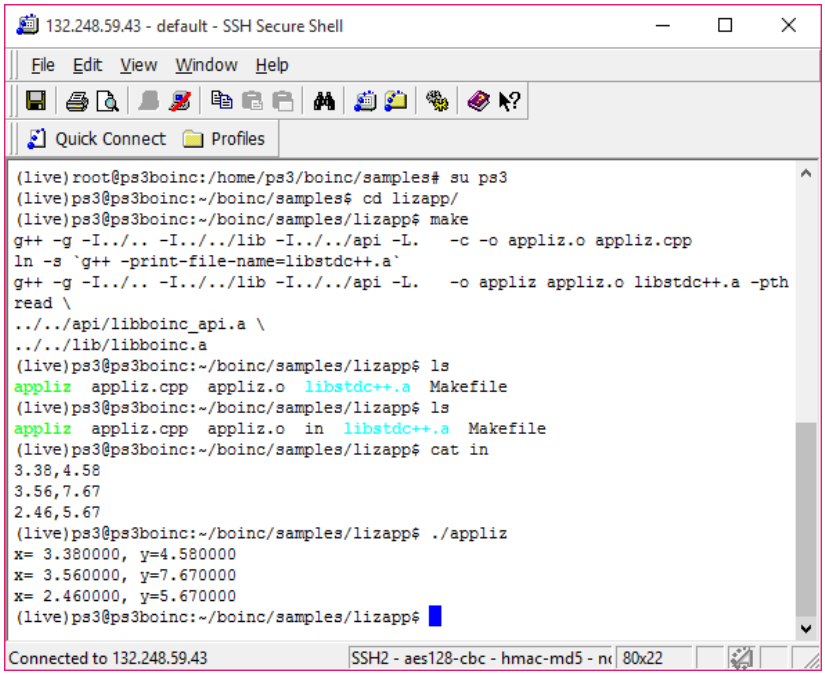


Figura 4.18 Compilación de la aplicación de trabajo de forma remota en un PlayStation3

4.4 GENERADOR DE TRABAJO

Un generador de trabajo crea los trabajos y sus archivos de entrada asociados. Requiere de las librerías boinc_db.h y backend_lib.h[64]. Un generador de trabajo se compone de cuatro funciones:

La función make_job() que se encarga de crear la unidad de trabajo, y de acuerdo a los requerimientos del proyecto puede crearse un archivo de trabajo para esa unidad de trabajo con un nombre único que se almacenará directamente en la carpeta download; en el siguiente código se muestra la función make_job para la aplicación de prueba provista en el paquete de software de BOINC.

```
int make_job() {
    DB_WORKUNIT wu;
    char name[256], path[MAXPATHLEN];
    const char* infiles[1];
    int retval;

  // make a unique name (for the job and its input file)
  //
    sprintf(name, "%s_%d_%d", app_name, start_time, seqno++);
```

```
    // Create the input file.
  // Put it at the right place in the download dir hierarchy
  //
    retval = config.download_path(name, path);
    if (retval) return retval;
    FILE* f = fopen(path, "w");
    if (!f) return ERR_FOPEN;
    fprintf(f, "This is the input file for job %s", name);
    fclose(f);

  // Fill in the job parameters
    //
    wu.clear();
    wu.appid = app.id;
    safe_strcpy(wu.name, name);
    wu.rsc_fpops_est = 1e12;
    wu.rsc_fpops_bound = 1e14;
    wu.rsc_memory_bound = 1e8;
    wu.rsc_disk_bound = 1e8;
    wu.delay_bound = 86400;
    wu.min_quorum = REPLICATION_FACTOR;
    wu.target_nresults = REPLICATION_FACTOR;
    wu.max_error_results = REPLICATION_FACTOR*4;
    wu.max_total_results = REPLICATION_FACTOR*8;
    wu.max_success_results = REPLICATION_FACTOR*4;
    infiles[0] = name;

    // Register the job with BOINC
    //
    sprintf(path, "templates/%s", out_template_file);
    return create_work(
        wu,
        in_template,
        path,
        config.project_path(path),
        infiles,
        1,
        config
    );
}
```

Dado que es un archivo de trabajo para la aplicación de prueba, el contenido del archivo será una cadena de caracteres que se convertirán en mayúsculas al ser ejecutado el trabajo, se configuran los parámetros para el trabajo.

La función main_loop() crea n unidades de trabajo con la configuración definida en make_job(), lo primero que hace es verificar que los demonios del servidor esten

activos, de ser así, revisa la base de datos del proyecto para obtener el número de trabajos sin enviar. Una vez obtenidos estos datos, crea las n unidades de trabajo.

```
void main_loop() {
    int retval;

    while (1) {
        check_stop_daemons();
        long n;
        retval = count_unsent_results(n, app.id);
        if (retval) {
            log_messages.printf(MSG_CRITICAL,
                "count_unsent_jobs() failed: %s\n", boincerror(retval)
            );
            exit(retval);
        }
        if (n > CUSHION) {
            daemon_sleep(10);
        } else {
            int njobs = (CUSHION-n)/REPLICATION_FACTOR;
            log_messages.printf(MSG_DEBUG,
                "Making %d jobs\n", njobs
            );
            for (int i=0; i<njobs; i++) {
                retval = make_job();
                if (retval) {
                    log_messages.printf(MSG_CRITICAL,
                        "can't make job: %s\n", boincerror(retval)
                    );
                    exit(retval);
                }
            }
            // Wait for the transitioner to create instances
            // of the jobs we just created.
            // Otherwise we'll create too many jobs.
            //
            double now = dtime();
            while (1) {
                daemon_sleep(5);
                double x;
                retval = min_transition_time(x);
                if (retval) {
                    log_messages.printf(MSG_CRITICAL,
                        "min_transition_time failed: %s\n",
                         boincerror(retval)
                    );
                    exit(retval);
```

```
                }
                if (x > now) break;
            }
        }
    }
}
```

La función usage solo se encarga de mostrar las opciones de linea de comandos que puede recibir el generador de trabajo.

```
void usage(char *name) {
    fprintf(stderr, "This is an example BOINC work generator.\n"
        "This work generator has the following properties\n"
        "(you may need to change some or all of these):\n"
        "  It attempts to maintain a \"cushion\" of 100 unsent job
           instances.\n"
        "  (your app may not work this way; e.g. you might create work in
            batches)\n"
        "- Creates work for the application \"example_app\".\n"
        "- Creates a new input file for each job;\n"
        "  the file (and the workunit names) contain a timestamp\n"
        "  and sequence number, so that they're unique.\n\n"
        "Usage: %s [OPTION]...\n\n"
        "Options:\n"
        "  [ --app X                Application name (default:
            example_app)\n"
        "  [ --in_template_file     Input template (default:
            example_app_in)\n"
        "  [ --out_template_file    Output template (default:
            example_app_out)\n"
        "  [ -d X ]                 Sets debug level to X.\n"
        "  [ -h | --help ]          Shows this help text.\n"
        "  [ -v | --version ]       Shows version information.\n",
        name
    );
}
```

La función main() se encargará de recibir los argumentos de línea de comandos, si son proporcionados, después, lee el archivo de configuración config.xml, inicia la comunicación con la base de datos, verifica que exista una versión de la aplicación y lee los archivos de configuración de las unidades de trabajo, una vez obtenidos todos estos requisitos se ejecuta la aplicación main_loop().[72]

```
int main(int argc, char** argv) {
    int i, retval;
    char buf[256];
```

```
    for (i=1; i<argc; i++) {
        if (is_arg(argv[i], "d")) {
            ...
            //En esta parte del código se leen los argumentos
            //de línea de comandos
        }
    }
    retval = config.parse_file();
    if (retval) {
        log_messages.printf(MSG_CRITICAL,
            "Can't parse config.xml: %s\n", boincerror(retval)
        );
        exit(1);
    }
    retval = boinc_db.open(
        config.db_name, config.db_host, config.db_user, config.db_passwd
    );
    if (retval) {
        log_messages.printf(MSG_CRITICAL, "can't open db\n");
        exit(1);
    }
    snprintf(buf, sizeof(buf), "where name='%s'", app_name);
    if (app.lookup(buf)) {
        log_messages.printf(MSG_CRITICAL, "can't find app %s\n",
        app_name);
        exit(1);
    }

    snprintf(buf, sizeof(buf), "templates/%s", in_template_file);
    if(read_file_malloc(config.project_path(buf), in_template)) {
        log_messages.printf(MSG_CRITICAL, "can't read input template
        %s\n", buf);
        exit(1);
    }
    start_time = time(0);
    seqno = 0;
    log_messages.printf(MSG_NORMAL, "Starting\n");
    main_loop();
}
```

4.5 VALIDADOR

Decide si los trabajos (jobs) completados son “válidos” y de ser así concede los créditos correspondientes al trabajo realizado. Para construir un validador se debe programar tres funciones:

```
int init_result(RESULT&, void*&)
```

Se encarga de leer los resultados contenidos en los archivos de salida que regresan los clientes y manipularlos para compararlos.

```
int compare_results(RESULT&, void*, RESULT const&, void*, bool&
match)
```

Se encarga de comparar dos resultados, en esta función se puede codificar una definición propia de igualdad, debido a que diferentes procesadores manejan de manera distinta los números de punto flotante.

```
int cleanup_result(RESULT const&, void*)
```

Se encarga de liberar el espacio en memoria de cada resultado después de realizar el proceso de validación.

El archivo con el código fuente de estas funciones se compila junto con el archivo validator.cpp en el que se llama a estas funciones para realizar la validación.

4.6 ASIMILADOR

El asimilador se encarga de procesar los resultados una vez que se ha decidido si son válidos o no, ya sea que solo se almacenen en la base de datos del proyecto o que se realice otro proceso con ellos.
Para crear un asimilado se debe programar la función

```
int assimilate_handler(
    WORKUNIT& wu, vector<RESULT>& results, RESULT& canonical_result
);
```

De acuerdo con la página oficial de BOINC, esta función se llama cuando:

- ✓ La unidad de trabajo tienen una máscara de error diferente de cero. En este caso el manejador puede escribir un mensaje en una bitácora o enviar un e-mail al administrador del proyecto.

- ✓ La unidad de trabajo tiene un resultado canónico, en ese caso wu.canonical_resultid será diferente de cero y canonical_result contendrá el resultado canónico.

En ambos casos el vector de resultados será populado con todos los resultados de todas las unidades de trabajo (incluyendo los no exitosos y los no enviados). Los valores de retorno de la función assimilate_handler() son:

- ✓ 0: exito. La unidad de trabajo será marcada como asimilada.
- ✓ DEFER_ASSIMILATION: La unidad de trabajo será procesada de nuevo cuando otra instancia termine. Se utiliza para aplicaciones que requieren todos los resultados completos.
- ✓ Otros valores diferentes de cero: El asimilador registrara un mensaje de error y terminará su ejecución

Los programas daemon se deben llamar desde el archivo config.xml de la siguiente forma[73]:

```
<daemon>
      <cmd> programa_daemon [argumentos] </cmd>
</daemon>
```

CAPÍTULO 5: IMPLEMENTACION DE PROYECTOS SOBRE LA PLATAFORMA BOINC

A continuación, se presenta el desarrollo de dos proyectos desarrollados para probar las capacidades de BOINC, el primero es Serpent, una aplicación de la que se cuenta con el código fuente, sin embargo, debido a la complejidad del algoritmo, y a que es un software del que constantemente se proporcionan actualizaciones, he optado por emplear una implementación con la aplicación Wrapper, que permite la portación de una aplicación a BOINC sin necesidad de modificar el código fuente.

El segundo proyecto pretende obtener la constante de Euler con la mayor precisión y con el mayor número de dígitos posible, empleando la definición de la constante como un límite al infinito, y empleando el desarrollo por binomio de Newton de dicha definición.

5.1 SERPENT EN BOINC

Serpent es un código tridimensional de energía continua basado en el método de Monte Carlo, que se emplea para la realización de cálculos y simulación de fenómenos físicos de reactores nucleares; fue desarrollado en 2004 por el Centro de Investigaciones Técnicas VTT en Finlandia.

Se emplea en cálculos de homogenización espacial y generación de constantes de grupo para cálculos determinísticos de simulación de reactores, en el estudio de ciclos de combustible, incluyendo cálculos detallados de quemado a nivel de ensamblaje, en la validación de códigos de transporte de retícula determinístico, en el modelado de reactores de investigación de núcleo completo, SMRs (Small Modular Reactors), y otros sistemas estrechamente acoplados, así como para la enseñanza y demostración de fenómenos de física de reactores[74][75].

5.1.1 REQUERIMIENTOS DEL SISTEMA

De acuerdo a la información proporcionada en la página web del proyecto, Serpent requiere de sistemas de 64 bits, ya sea en sistemas operativos GNU/Linux o MAC OS. Y dependiendo del tipo de cálculo a realizar puede requerir de hasta 10 GB de

memoria RAM. Es inestable en sistemas operativos Windows, aunque puede compilarse con la ayuda de herramientas de plataforma cruzada[75].

5.1.2 CONFIGURACIÓN DE SERPENT.

Se descomprime el código fuente, y se debe modificar el Makefile para quitar la configuración de las librerías gd como se muestra en la figura 5.1[75].

```
########################################################################
######

# GD graphics library:

#LDFLAGS += -lgd

# Compile with the following option if GD library is not available:

CFLAGS += -DNO_GFX_MODE

########################################################################
```

Figura 5.1 Makefile de Serpent

Junto con el software Serpent y las librerías se incluye el script xdirconverter.pl que se encarga de generar el archivo xsdata con las rutas de las librerías ACE, primero es necesario darle permisos de ejecución al script xsdirconvert.pl como se muestra en la figura 5.2.

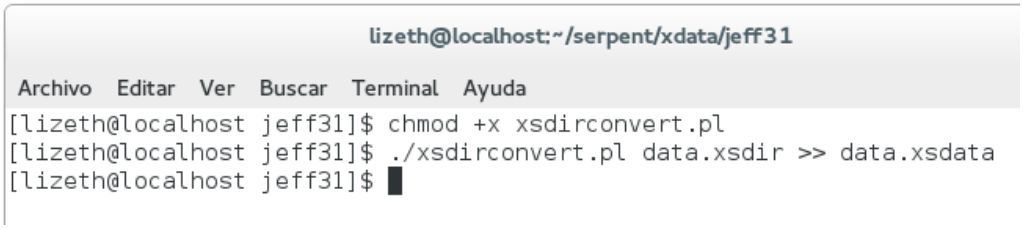


Figura 5.2 Ejecución del script xsdirconvert.pl

Una vez hechas las configuraciones, se ejecuta Serpent con un archivo de entrada de ejemplo. Serpent recibe el nombre del archivo de entrada por línea de comandos, los archivos de entrada no tiene un formato especifico, solo utilizan distintas etiquetas para referirse a los componentes a partir de los cuales se realizarán los cálculos, lo que no cambia es que en alguna parte del archivo, se debe incluir la ruta del archivo xsdata que contiene las rutas de los archivos ACE y además las rutas a librerías especiales, por ejemplo librerías para cálculos de fusión y fisión.

Una vez configurado y probado Serpent con éxito, se comienzan las pruebas para correr Serpent sobre BOINC.

5.1.3 BOINC WRAPPER Y SERPENT

Cuando ya se tiene una aplicación compilada y no se desea, o no es posible adaptar el código fuente para que trabaje con las librerías de BOINC, se puede utilizar la aplicación llamada wrapper, esta aplicación ejecuta las aplicaciones como subprocesos y se encarga de manejar la comunicación con el cliente de BOINC. Con el código fuente de BOINC se incluye el código fuente de la aplicación wrapper, o se puede descargar una versión compilada de esta aplicación de la página http://boinc.berkeley.edu/trac/wiki/WrapperApp.

Con el fin de no modificar el código fuente de Serpent, se consideró que la mejor forma de abordar el problema inicialmente es empleando el programa wrapper de BOINC.

Para la implementación de la aplicación wrapper se requiere:

- La aplicación wrapper de BOINC.
- La aplicación de trabajo, es este caso Serpent.
- Un archivo de configuración de la versión de la aplicación, version.xml.
- Un archivo de descripción de trabajo para la aplicación wrapper, job.xml.

El archivo version.xml se utiliza para describir una versión de una aplicación en un proyecto, normalmente contiene el nombre físico de la aplicación en el servidor, y el nombre lógico, que es el nombre que tendrá en las maquinas cliente, además se configura la aplicación como aplicación principal. En este enfoque se configura la aplicación wrapper como aplicación principal, y después se configura la aplicación de trabajo y el archivo de descripción de trabajo de la aplicación wrapper. El contenido del archivo version.xml es el siguiente[61][62]:

```
<version>
<file>
      <physical_name>wrapper</physical_name>
      <main_program/>
</file>
<file>
      <physical_name>sss2_1.0_x86_64-pc-linux-gnu</physical_name>
      <logical_name>sss2</logical_name>
</file>
<file>
      <physical_name>sss2_job_1.0.xml</physical_name>
      <logical_name>job.xml</logical_name>
</file>
```

```
</version>
```

El archivo de descripción de trabajo especifica el nombre de la aplicación de trabajo a ejecutar, opciones por línea de comandos que necesite recibir la aplicación de trabajo, así como el manejo de los archivos de manejo de errores y mensajes del cliente de BOINC y los archivos de entrada y salida en caso de ser archivos comprimidos. Para las pruebas preliminares se utilizará un archivo muy sencillo, que solo contiene el nombre de la aplicación y como comandos el nombre del archivo de entrada, ya que Serpent recibe el nombre del archivo de entrada por línea de comandos [62].

```
<job_desc>
      <task>
            <application>sss2</application>
            <command_line>in</command_line>
      </task>
</job_desc>
```

5.1.4 PRUEBA STANDALONE.

Para la prueba standalone se requiere únicamente la aplicación wrapper, la aplicación de trabajo, el archivo de trabajo de la aplicación wrapper con nombre lógico job.xml, y el archivo de entrada, también con nombre lógico in como se muestra en la figura 5.3.

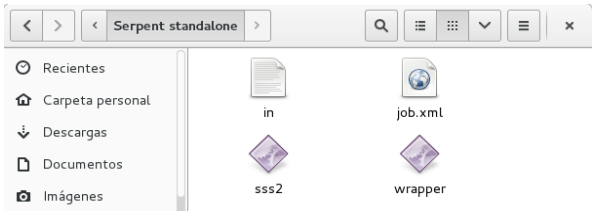


Figura 5.3 Archivos para la prueba standalone

A continuación se muestra una captura de pantalla de una parte del archivo de entrada en la figura 5.4.

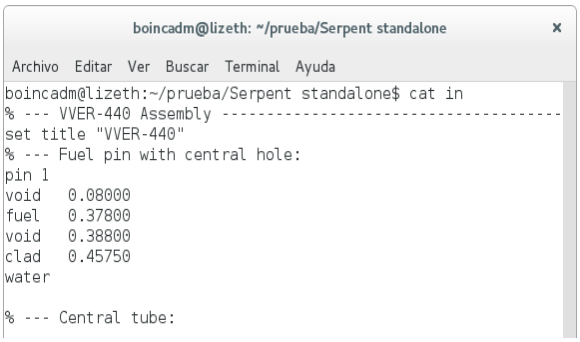


Figura 5.4 Archivo de entrada

En la figura 5.5 se muestra la sección del archivo de entrada que contiene la configuración de las rutas de las librerías necesarias para que Serpent realice los cálculos requeridos.

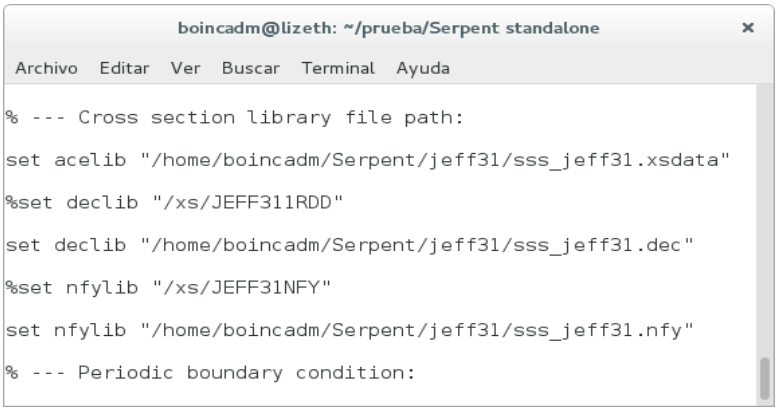


Figura 5.5 Referencia a las rutas en el archivo de entrada

De acuerdo con la documentación de Serpent es posible dividir la entrada del programa en varios archivos, por lo que es necesario emplear algún método para enviar más de un archivo de entrada a los clientes. A mi consideración la forma más eficiente de enviar los archivos es a través de un archivo comprimido en formato zip, el archivo recibirá el nombre de in.zip; la aplicación wrapper de BOINC está diseñada para manipular archivos comprimidos en este formato, solo es necesario añadir al archivo de configuración del trabajo las siguientes etiquetas[62]:

```
<unzip_input>
    <zipfilename>in.zip</zipfilename>
</unzip_input>
```

Se ejecuta la aplicación wrapper, que a su vez ejecuta la aplicación de trabajo Serpent.

```
boincadm@lizeth:~/prueba/Serpent standalone$ ./wrapper
12:10:17 (11439): wrapper (7.7.26016): starting

  _                    .-=-.           .-=-.          .-==-.
 { }      __         .' O o '.       .' O o '.       /  -<' )--
<
 { }    .' O'.      / o .-. O \     / o .-. O \     /  .---`
 { }   / .-. o\   /O  /   \  o\   /O  /   \  o\   /O /
  \ `-` /   \ O`-'o  /     \  O`-'o  /     \  O`-`o /
   `-.-`     '.____.'       `._____.'       `.____.'


Serpent 2 beta

A Continuous-energy Monte Carlo Reactor Physics Burnup Calcula
tion Code

 - Version 2.1.0 (February 3, 2012) -- Contact: Jaakko.Leppane
n@vtt.fi
```

Figura 5.6 Ejecución de Serpent a través de la aplicación wrapper

Se generan los archivos de salida de la aplicación Serpent (in.out, in.seed y los resultados en formato de matlab) junto con los logs de BOINC generados por la aplicación wrapper, esto puede observarse en la figura 5.7.

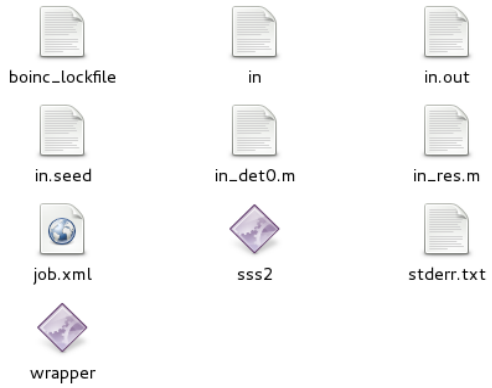


Figura 5.7 Archivos generados después de la ejecución de Serpent

Como se puede observar en la imagen, Serpent genera varios archivos de entrada, por lo que también será necesario enviar los archivos en una carpeta zip, Wrapper provee soporte para comprimir los archivos una vez que termino a ejecución de la aplicación de trabajo, se añaden las siguientes etiquetas al archivo de descripción de trabajo del wrapper:

```
<zipfilename>out.zip</zipfilename>
    <filename>[in][a-z\.\_]*</filename>
</zip_output>
```

Se emplea una expresión regular para indicar que archivos serán comprimidos en el archivo out.zip.

5.1.5 CREACIÓN DE UN PROYECTO VACÍO PARA SERPENT.

El primer paso es crear un proyecto vacío con el script make_project, que creará los componentes del proyecto, esto se muestra en la figura 5.8:

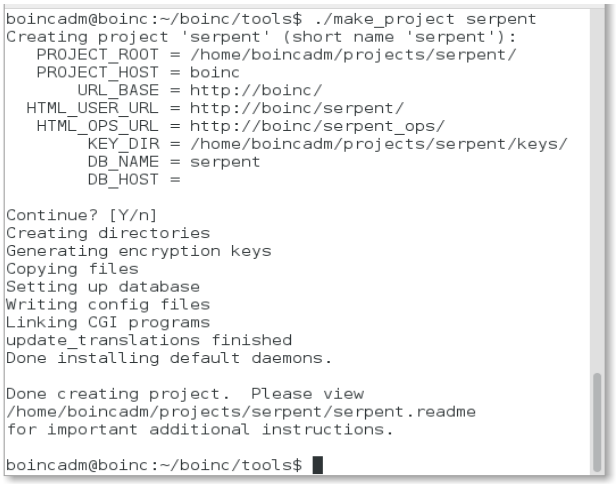


Figura 5.8 Salida del script make_project

Dado que es un proyecto vacío se debe crear la estructura de directorios para las aplicaciones, la carpeta principal del proyecto tiene el nombre de Serpent, la estructura quedará de la siguiente forma, solo para la versión de la aplicación para Linux de 64 bits en procesadores Intel:

```
Serpent/
      apps/
            sss2/
                  1.0/
                        x86_64-pc-linux-gnu/
```

En esta última carpeta se copia la aplicación wrapper, se crea el archivo version.xml, se copia la aplicación de trabajo Serpent, el nombre original del ejecutable es sss2, pero se renombrará con el formato especificado para las aplicaciones de trabajo para BOINC, por lo que el nuevo nombre será: sss2_1.0_x86_64-pc-linux-gnu, este nombre será el nombre físico de la aplicación, el nombre lógico será el original sss2. Además, en esta carpeta debe estar el archivo de configuración de trabajo de la aplicación wrapper, cuyo nombre físico será sss2_job_1.0.xml.

Se configura la página web en el servidor apache, añadiendo la ruta del archivo de configuración del proyecto al archivo apache2.conf, como se muestra en la figura 5.9.

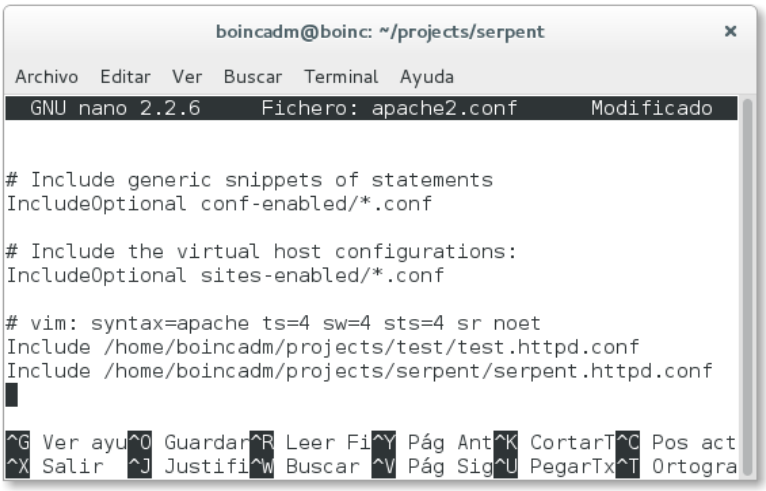


Figura 5.9 Configuración de Apache

Se añade el proyecto a cron como se muestra en la figura 5.10.

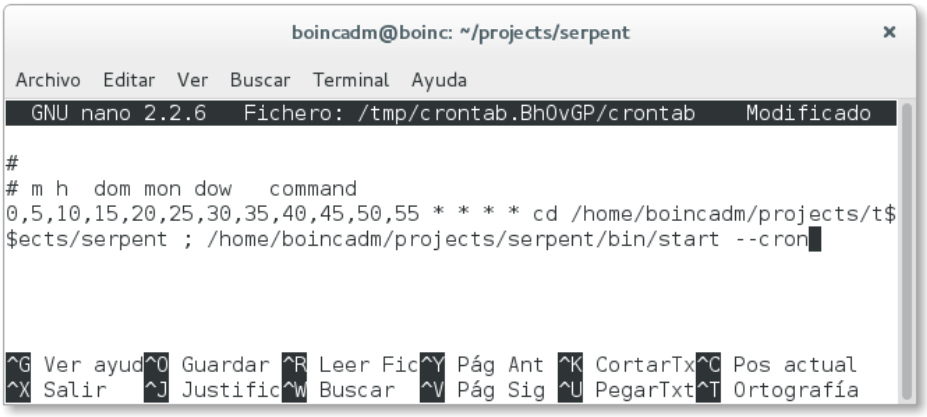


Figura 5.10 Configuración de cron

Se modifica el archivo project.xml para configurar las versiones de las aplicaciones existentes y configurar el nombre de la aplicación.

El contenido del archivo project.xml es el siguiente:

```
<boinc>
      <platform>
            <name>x86_64-pc-linux-gnu</name>
            <user_friendly_name>Linux running on an AMD x86_64 or Intel
            EM64T CPU</user_friendly_name>
      </platform>
      <app>
```

```
            <name>sss2</name>
            <user_friendly_name>Serpent sobre BOINC</user_friendly_name>
        </app>
</boinc>
```

Se inicializa la base de datos y se instala la aplicación con los comandos bin/xadd que se muestra en la figura 5.11 y bin/update_versions que se muestra en la figura 5.12:

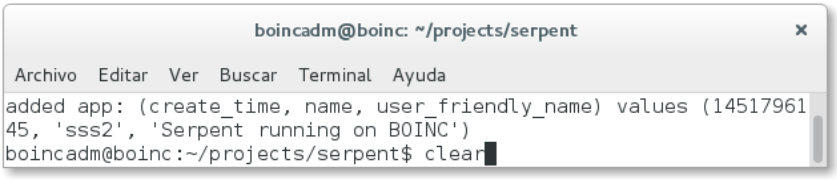


Figura 5.11 Salida del comando xadd

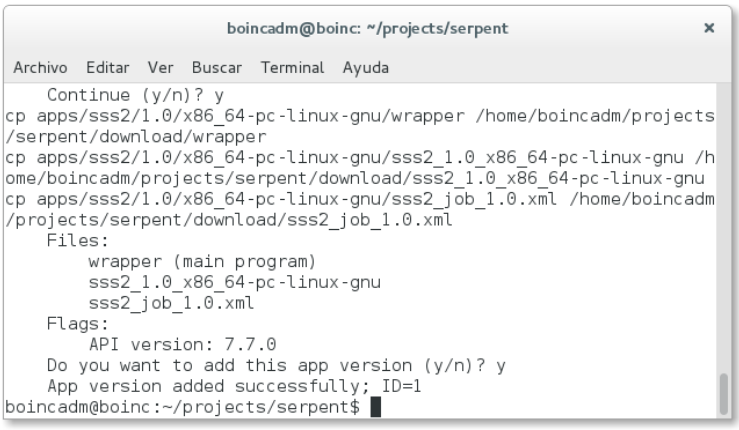


Figura 5.12 Salida del comando update_versions

Se inicia el proyecto con el comando bin/start como se muestra en la figura 5.13.

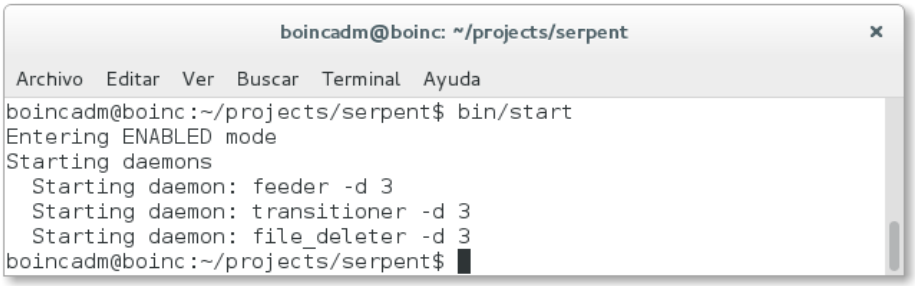


Figura 5.13 Inicialización del proyecto

Se crean los archivos de descripción de las unidades de trabajo y de los archivos de salida y se almacenan en la carpeta templates como se muestra en la figura 5.14.

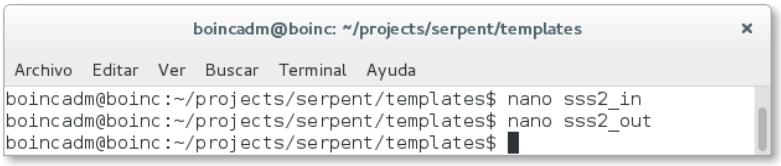


Figura 5.14 Creación de las plantillas para las unidades de trabajo

El contenido del archivo sss2_in es el siguiente:

```
<file_info>
      <number>0</number>
</file_info>
<workunit>
      <file_ref>
      <file_number>0</file_number>
      <open_name>in</open_name>
      <copy_file/>
      </file_ref>
      <rsc_fpops_bound>1e12</rsc_fpops_bound>
      <rsc_fpops_est>1e14</rsc_fpops_est>
</workunit>
```

El contenido del archivo sss2_out es el siguiente:

```
<file_info>
      <name><OUTFILE_0/></name>
      <generated_locally/>
      <upload_when_present/>
      <max_nbytes>5000000</max_nbytes>
      <url><UPLOAD_URL/></url>
</file_info>
<result>
      <file_ref>
      <file_name><OUTFILE_0/></file_name>
      <open_name>out</open_name>
      <copy_file/>
      </file_ref>
</result>
```

Para las primeras pruebas se sube un archivo de entrada manualmente con el comando bin/stage_file, en este caso el archivo se encontraba en la carpeta de la prueba standalone.

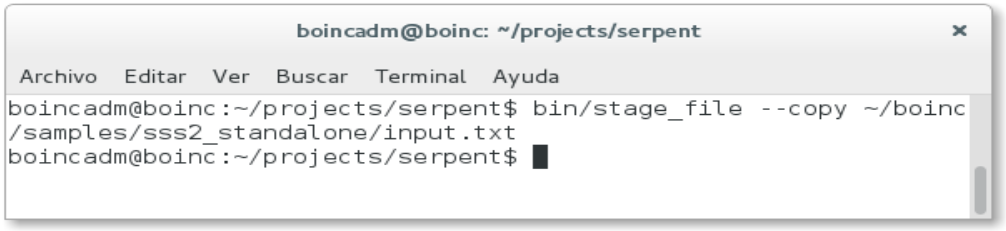


Figura 5.15 Subida de un archivo de entrada con stage_file

5.1.6 PRUEBA CON UN CLIENTE DENTRO DEL SERVIDOR

Dentro del mismo servidor se crea una carpeta para pruebas con el cliente y se copia el ejecutable del cliente boinc_client y la herramienta de línea de comandos del cliente boinccmd.

Figura 5.16 Conexión del cliente al proyecto

Se crea una cuenta en el proyecto para obtener una clave de autenticación. Se ejecuta el cliente boinc_client, y a través de la herramienta de línea de comandos se conecta el cliente al proyecto como se muestra en la figura 5.16.

5.1.7 CONEXIÓN DEL PROYECTO AL CLIENTE

Para las pruebas preliminares se crea una unidad de trabajo manualmente con el comando bin/create_work desde la carpeta bin del proyecto como se muestra en la figura 5.17.

Figura 5.17 Creación de una unidad de trabajo manualmente desde el servidor

Y el cliente comienza a realizar el trabajo, y se crean los archivos de salida de la unidad de trabajo. En los clientes se crean carpetas llamadas slot que contienen los archivos de salida de las aplicaciones, el contenido de las carpetas que se crean en slot es temporal, solo existen mientras se ejecuta el trabajo y después de destruyen, el contenido del archivo slot/0 se muestra en la figura 5.18.

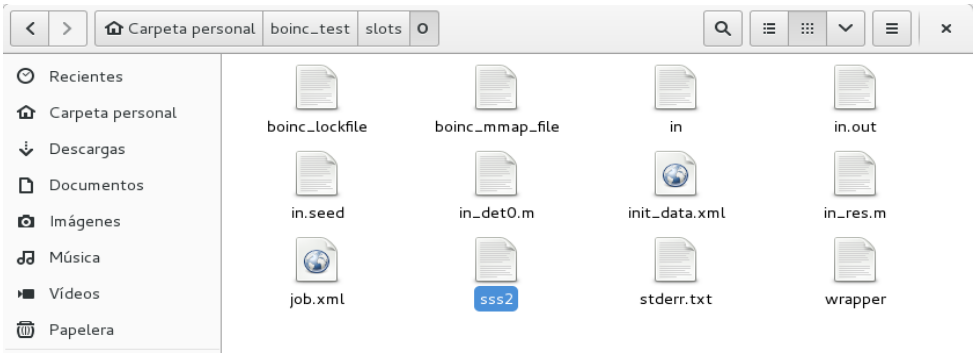


Figura 5.18 Contenido de uno de los directorios creados por el cliente para una unidad de trabajo

En las primeras pruebas las librerías ACE se copiaron manualmente a una carpeta en la computadora cliente y en la computadora servidor, en las que se realizaron las pruebas. Debido a que algunos de estos archivos son muy pesados, en conjunto las librerías pesan aproximadamente 5GB, y no es muy conveniente enviarlos en un archivo zip, ya que de comprimirse, no se reduce considerablemente el tamaño de la carpeta comprimida y se requiere de un gran ancho de banda para su descarga.

La solución empleada para enviar las librerías a los clientes es enviarlas junto con la aplicación de trabajo y sus archivos relacionados, de tal forma que se envían los archivos uno a uno a los clientes, se calcula un MD5 por cada archivo, lo que permite tener un mejor control de la integridad de los archivos, y en caso de que se suspenda la descarga, se puede continuar desde el archivo faltante y no recomenzar la descarga de un archivo zip de más de 4GB, ya que el archivo que contiene la librería más grande pesa aproximadamente 250MB.

Para este fin se copian las librerías a la carpeta en donde se encuentra la aplicación, en el directorio apps del proyecto, y se añaden las referencias al archivo versión.xml. Además he creado un script que se encarga de crear las carpetas jeff31 y acedata en donde originalmente se encuentran estos archivos y los guarda en la carpeta con el nombre del servidor que se crea en los clientes, para sistemas GNU/Linux está en la ruta /var/lib/boinc, las librerías se muestran en la figura 5.19.

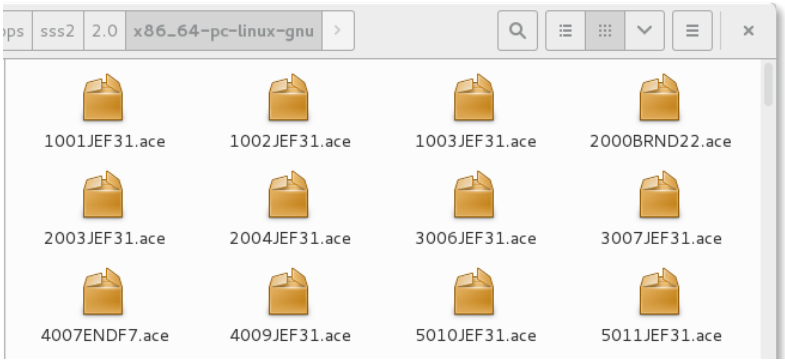


Figura 5.19 Librerías ACE

Una vez resuelto el problema de las librerías se desarrolla un generador de trabajo, debido a que los archivos de entrada son específicos para los cálculos a realizar, no es posible automatizar su creación, así que la mejor opción es desarrollar un generador de trabajo que se encargue de leer los archivos de entrada desde una carpeta especial, de tal manera que solo será necesario subir los archivos de entrada a dicha carpeta para que el generador de trabajo los lea y cree la unidad de trabajo.

Para este fín se emplea la clase DirScanner de la librería filesys.h provista con el código fuente de BOINC, se implementa el código para escanear el directorio en el que se almacenarán los archivos de entrada en la función main_loop, el cual se muestra a continuación, la implementación se muestra en la figura 5.20:

```
DirScanner dirscan(ARCHIVOS);
// check if there are files in INPUT_FOLDER
if (dirscan.scan(file)) {
    // files found, now create work for them
    int njobs = (CUSHION-n)/REPLICATION_FACTOR;
    log_messages.printf(MSG_DEBUG,
        "Haciendo %d trabajos\n", njobs
    );
    for (int i=0; i<njobs; i++) {
        retval = make_job(file.c_str());
        if (retval) {
            log_messages.printf(MSG_CRITICAL,
                "No se pudo hacer el trabajo: %d\n", retval
            );
            exit(retval);
        }
        if (!dirscan.scan(file)) break; // no more files but CUSHION not yet reached
    }
```

Figura 5.20 Escaneo de directorios en el generador de trabajo

Por cada archivo encontrado, se crea una unidad de trabajo; cuando se encuentra un archivo, se obtiene el nombre del mismo, y se pasa como parámetro a la función make_job, en la que se implementó el código que se encarga de mover el archivo de entrada al directorio de descarga, en la figura 5.21 se muestra la porción de código que realiza este trabajo:

```
    // make a unique name and move input file into download dir
    sprintf(wu_name, "sss2_%d_%d", start_time, seqno++); // TODO: change this
rate the workunitname using the input file name
    log_messages.printf(MSG_DEBUG,
            "Archivo %s encontrado, creando trabajo %s\n", filename, wu_name
    );
    retval = config.download_path(filename, dst_path);
    if (retval) return retval;
    sprintf(src_path, "%s/%s", ARCHIVOS, filename);
    log_messages.printf(MSG_DEBUG,
            "Moviendo archivo %s a %s\n", src_path, dst_path
    );
    retval = rename(src_path, dst_path);
    if (retval) {
        log_messages.printf(MSG_CRITICAL,
            "rename: %d, errno is %d\n", retval, errno
        );
        return retval;
:
```

Figura 5.21 Creación de unidades de trabajo en el generador de trabajo

5.2 NÚMERO DE EULER

El número e o constante de Euler es un número trascendental cuyo valor aproximado es de 2.718281828459045... Existen varias formas de definir el numero e, en el presente trabajo se utilizará la definición de e como:

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^{n} \quad (1)$$

De la ecuación anterior se puede deducir que mientras mayor sea el valor de e mejor será la aproximación al valor de e. Para la implementación de el cálculo del número de Euler de forma distribuida, empleando el binomio de Newton se desarrollará el binomio:

$$\left(1+\frac{1}{n}\right)^{n} \quad (2)$$

El binomio de Newton que nos permite expresar la enésima potencia de un binomio como un polinomio. Cada uno de los términos del polinomio obtenido se convertirá en una unidad de trabajo que será enviada a cada uno de los clientes.

El teorema del binomio expresado en términos combinatorios se define con la fórmula:

$$(a+b)^n = \sum_{m=0}^{n} \binom{n}{m} a^{n-m} b^m \quad (3)$$

A continuación se muestra el desarrollo de la ecuación 3 para diferentes valores de n:

$$(a+b)^2 = a^2 + 2ab + b^2$$
$$(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$$
$$(a+b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$$
$$(a+b)^5 = a^5 + 5a^4b + 10a^3b^2 + 10a^2b^3 + 5ab^4 + b^5$$

De este desarrollo se observan las siguientes características:

Del desarrollo de $(a+b)^n$ se obtendrá un polinomio de n+1 términos.
Las potencias de a comienzan en n y van disminuyendo en una unidad en cada termino hasta llegar a 0, mientras que las potencias de b comienzan en cero van aumentando en cada termino hasta llegar a n, además la suma de los exponentes de a y b siempre será igual a n.

El coeficiente del primer término es uno, y para los términos subsecuentes será igual al producto del coeficiente del termino anterior por el exponente de a dividido entre el número que indica el orden de ese término, considerando el 0 para el primer término del polinomio.

Por lo tanto, el desarrollo del binomio $(a+b)^n$ será:

$$(a+b)^n = a^n + \frac{n}{1}a^{n-1}b + \frac{n(n-1)}{1(2)}a^{n-2}b^2 + \frac{n(n-1)(n-2)}{1(2)(3)}a^{n-3}b^3 + \ldots + b^n \quad (4)$$

Se sabe que el coeficiente determina el número de veces que se repite un término al desarrollar la multiplicación del binomio por si mismo n veces, esto puede expresarse de forma combinatoria como:

$$(a+b)^n = \binom{n}{0}a^n + \binom{n}{1}a^{n-1}b + \binom{n}{2}a^{n-2}b^2 + \binom{n}{3}a^{n-3}b^3 + \ldots + \binom{n}{n}b^n \quad (5)$$

Dado lo anterior para calcular el k-ésimo término del polinomio, se obtiene:

$$T_k = \binom{n}{k-1} a^{n-(k-1)} b^{k-1} \quad (6)$$

Sustituyendo a = 1 y b = 1/n en la ecuación 6:

$$T_k = \binom{n}{k-1} 1^{n-(k-1)} \left[\frac{1}{n}\right]^{k-1} \quad (7)$$

Debido a que 1 elevado a cualquier potencia siempre será 1, la ecuación 7 se reduce a :

$$T_k = \binom{n}{k-1} \left[\frac{1}{n}\right]^{k-1} \quad (8)$$

Para el cálculo del término combinatorio se emplea la fórmula:

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} \quad (9)$$

En donde $n \geq m$.[76][77]

5.2.1 MANEJO DE NÚMEROS DE PUNTO FLOTANTE

El manejo de números de punto flotante, tanto en hardware como en software esta definido en el estándar 754 de la IEEE, de acuerdo al cual, un número de punto flotante esta compuesto por un signo, una mantisa y un exponente, el estándar define dos tipos de precisión, la precisión simple de 32 bits y la doble de 64 bits.

Para la precisión simple, se emplean 8 bits para representar al exponente y 23 bits para representar la mantisa. Para la precisión doble se emplean 11 bits para el exponente y 52 bits para la mantisa, en ambos tipos de precisión se emplea un bit para determinar el signo del numero representado. En la siguiente tabla se muestra la distribución de los bits para la representación de números de punto flotante en precisión doble y en precisión simple.

Tabla 5.1 Presición de los números de punto flotante

	Signo	Exponente	Mantisa	Número mínimo representable	Número máximo representable
Precisión simple	1[31]	8[30 - 23]	23[22-0]	$1.175x10^{-38}$	$3.403x10^{38}$
Precisión doble	1[63]	11[62 - 52]	52[51 - 0]	$2.225x10^{-308}$	$1.798x10^{308}$

El exponente debe ser capaz de representar tanto valores positivos como negativos, por lo que se emplea un valor de sesgo que se suma al valor del exponente a representar antes de almacenarlo. Para precisión simple se emplea un sesgo de 127 y para precisión doble se emplea un sesgo de 1023.

La mantisa está normalizada con un bit implícito, es decir, se considera que el primer bit de la mantisa siempre es un 1, este 1, no se incluye al almacenar el número, sino que se añade al momento de utilizar el número almacenado, esto permite un bit mas de precisión.
Si los números que se intenta representar están fuera de los rangos descritos en la tabla se presentan errores de desbordamiento y subdesbordamiento.[78]

5.2.2 APLICACIÓN DE TRABAJO

Una vez realizado todo este análisis se puede desarrollar el algoritmo para codificar el programa cliente y el generador de trabajo que se encarga de crear una unidad de trabajo para cada termino del polinomio, además, se encargara de crear los archivos de entrada asociados a cada unidad de trabajo.
Empleando ciclos para las funciones se agiliza el cálculo de las potencias y los factoriales, a continuación se muestra la función que se encarga de calcular el factorial de un número:

```
long int factorial(int n) {
      long int fact;
      int i;
      fact = 1;
      if (n == 0 || n == 1) {
            fact = 1;
      }
      else {
            for (i = 1; i <= n; i++) {
                  fact *= i;
            }
      }
```

```
      return fact;
}
```

El siguiente es el código que calcula la n-esima potencia de un número:

```
long double potencia(double var, int exp) {
      int i;
      long double pot;
      pot = 1;
      if (exp != 0){
            for (i = 1; i <= exp; i++) {
                  pot *= var;
            }
      }
      return pot;
}
```

A partir de estas dos funciones se desarrollaron las funciones que calculan el coeficiente binario y el k-esimo término. El siguiente código corresponde a la función que calcula el término combinatorio o coeficiente binario del k-ésimo término:

```
long int coefBin(int n, int m) {
      long int coef;
      coef = factorial(n) / (factorial(m) * factorial(n - m));
      return coef;
}
```

A continuación se muestra el código de la función que calcula el k-ésimo término:

```
long double ktermino(int n, int k) {
      long double termino = 1;
      termino = coefBin(n, (k - 1)) * potencia((1 / n), (k - 1));
      return termino;
}
```

Al probar el algoritmo inicial implementado en c, noté que aun empleando variables double de doble precisión, llega un momento en que se comienza a presentar problemas de desbordamiento. Con tipos de dato double de doble precisión no podían alcanzarse valores de k mayores de 20 con n = 100.

Se necesita que el sistema sea capaz de manejar rangos muy grandes de valores, se realizó la implementación del algoritmo empleando librerías de precisión arbitraria.

Existen varias opciones, en cuanto a librerías de precisión arbitraria se refiere, en la tabla 5.2 se muestran algunas librerías de precisión arbitraria conocidas:

Tabla 5.2 Librerías de punto flotante

Librería	Licencia	Características	Observaciones
TTMath	BSD, de libre uso.	Permite realizar operaciones con variables de tipo entero, con signo y sin signo, y con variables de punto flotante.	En plataformas de 32 bits, el valor máximo que una variable de tipo entero puede alcanzar es $2^{(32*2)}$-1
MPIR	LGPL v3	Esta escrita en C y ensamblador. Para asegurar el mejor funcionamiento es necesario especificar el tipo de procesador que se utilizará, si la arquitectura no esta soportada se emplea la configuración genérica. El cálculo se hace lento al manejar números pequeños	Es una variante de GMP, esta escrita en ensamblador y C, tiene soporte para Windows. Requiere yasm para su compilación.
CLN	GPL	Provee soporte para números complejos, el manejo de memoria permite realizar cálculos rápidos con números muy pequeños, contiene un núcleo especial para algunos CPU's escrito en ensamblador.	Requiere gnu sed para su compilación. Para arquitecturas de 32 bits se puede realizar la conversión de variables int para valores < 2^{29} y => -2^{29} y unsigned int para valores < 2^{29}
Arbitrary precision package	Libre uso	Provee un conjunto amplio de funciones trigonometricas y logaritmicas y 4 tipos de redondeo.	Soporta números de hasta 4 billones de dígitos. Sin embargo, dependiendo de la arquitectura pueden presentarse problemas antes de alcanzar números de esa magnitud.
GMP	LGPL v3 y LGPL v2	El número máximo de dígitos que puede contener un número esta limitado por la capacidad de la memoria. Esta diseñada para trabajar de manera rápida con números pequeños y grandes.	Contiene un amplio conjunto de funciones para variables enteras, racionales y flotantes, sin embargo, se recomienda emplear MPFR para proyectos que emplearán

		Esta optimizada con lenguaje ensamblador para mejorar el funcionamiento en distintas arquitecturas de cpu. Esta orientado a sistemas unix, pero tambien puede emplearse en Windows.	variables de tipo flotante, ya que esta librería es mucho mas completa y esta especialmente dedicada para el uso de variables de punto flotante.

5.2.3 IMPLEMENTACIÓN DEL CÓDIGO UTILIZANDO MPFR

La librería MPFR es una librería diseñada en el lenguaje de programación C, para el manejo de variables de tipo flotante de precisión arbitraria con soporte para el redondeo correcto. La librería MPFR es mantenida por el INRIA (Instituto Nacional de Investigación en Informática y Automática) de Francia, y esta basada en GMP. Su objetivo principal es proveer una librería de punto flotante de precisión arbitraria eficiente y con una semántica bien definida basada en el estándar ANSI/IEEE 754. Tiene soporte para diversos lenguajes de programación, tales como PYTHON, RUBY, LISP, PERL, R y Java.
Las variables de precisión arbitraria empleando MPFR se declaran de la siguiente forma:

```
mpfr_t nombre_variable;
```

El segundo paso es utilizar alguna función de inicialización para indicar el tamaño en bites que utilizará dicha variable, en este caso se emplea la función mpfr_init2(), que recibe dos argumentos, el nombre de la variable a inicializar y el tamaño en bites que se reservará para la variable.

```
mpfr_init2(nombre_variable, tamaño);
```

MPFR provee varias funciones útiles, en el desarrollo de las funciones que calcularán el k-esimo término del binomio de Newton para el cálculo del número de Euler se emplearán las funciones básicas de multiplicación y división.

```
mpfr_mul(var_destino, multiplicando, multiplicador,
redondeo);
mpfr_div(var_destino, dividendo, divisor, redondeo);
```

Cuando se una variable de precisión arbitraria ya no es necesaria, se debe liberar el espacio en memoria que ocupa, esto se realiza empleando la función:

```
      mpfr_clear(nombre_variable);
```

Para que MPFR funcione, es necesario que la librería GMP este instalada también, ya que MPFR depende de GMP.[79]

Además de la implementación con MPFR, se hicieron cambios en el código con el fin de simplificar el algoritmo. Se sabe que la potencia a calcular es la k-esima potencia de 1/n, por lo que resulta mas conveniente implementarla como un ciclo de divisiones, como se muestra a continuación:

```
void potenciab(mpfr_t pot, mpfr_t base, unsigned long int exp){
      unsigned long int i;
      fprintf(stderr, "Calculando la potencia %lu de 1/", exp);
      mpfr_fprintf(stderr, "%.1Rf", base);
      fprintf(stderr,"\n");
      mpfr_set_d(pot, 1.0, MPFR_RNDD);

      if(exp != 0)
            for(i=1; i<=exp; i++){
                  mpfr_div(pot, pot, base, MPFR_RNDD);
            }
}
```

Los factoriales que se emplean en el cálculo del coeficiente binario, se calculan dentro de la misma función de tal forma que el código de dicha función es:

```
void coefBin(mpfr_t coef, unsigned long int n, unsigned long int
m){
      unsigned long int i;
      mpfr_t f;
      mpfr_init2(f,500);
      mpfr_set_d(f, 1.0, MPFR_RNDD);
      //Sabemos que n siempre será mayor que m y que (n-m)
      fprintf(stderr, "Calculando el coeficiente binario...\n");
      mpfr_set_d(coef, 1.0, MPFR_RNDD);
      if(n!=0)
                  for(i=1; i<=n; i++){
                          mpfr_mul_ui(f, f, i, MPFR_RNDU);

                        if(i == (n-m)|| i == m){
                              mpfr_div(coef, coef, f, MPFR_RNDD);
                        }
                  }
      mpfr_mul(coef, coef, f, MPFR_RNDD);
      mpfr_clear(f);
}
```

Finalmente se calcula el k-esimo término empleando la siguiente función:

```
void ktermino(unsigned long int ulint_k, unsigned long int ulint_n
){
      mpfr_t termino, bk, pbk, back;
      mpfr_init2(termino, 500);
      mpfr_init2(bk, 500);
      mpfr_init2(pbk, 500);
      mpfr_init2(back, 500);

      ulint_k -= 1;

      coefBin(termino, ulint_n, ulint_k);
      mpfr_set_ui(bk, ulint_n, MPFR_RNDD);

      potenciab(pbk, bk, ulint_k);

      mpfr_mul(termino, termino, pbk, MPFR_RNDD);
      fprintf(stderr, "\n\nEl k termino es: \n");
      mpfr_fprintf(stderr, "%.4096Rf", termino);
      mpfr_out_str(salida, 10, 8192, termino, MPFR_RNDD);
      fprintf(stderr, "\n");

      mpfr_clear(bk);
      mpfr_clear(pbk);
      mpfr_clear(termino);
}
```

Se emplean variables de 500 bits. Una vez implementado el algoritmo empleando la librería MPFR, se realiza la conexión con el API de BOINC para el manejo de los archivos de entrada y salida y la comunicación con el cliente. El código completo se encuentra en la sección anexo B.

5.2.4 GENERADOR DE TRABAJO

La aplicación de trabajo del proyecto se encarga de calcular el k-esimo de n términos en las máquinas cliente, para realizar dichos cálculos, recibe como argumentos, el valor de n y el valor de k, dado que el valor de n siempre es el mismo, y k puede tomar valores de 1 hasta n, es posible utilizar un generador de

trabajo que genere los archivos de entrada que se enviarán a la aplicación de trabajo de forma automática.
Se utilizó el código fuente del generador de trabajo provisto por BOINC para desarrollar un generador de trabajo propio. En la función make_job, se implementa la creación de los archivos de entrada, la función make_job recibirá como argumento el término a calcular para una determinada unidad de trabajo; en el siguiente código se muestra la creación del archivo de entrada:

```
  retval = config.download_path(name, path);
    if (retval) return retval;
    FILE* f = fopen(path, "w");
    if (!f) return ERR_FOPEN;
    fprintf(f, "%lu,%lu", ne, kterm);
  fclose(f);
```

La variable ne contiene el tamaño de n que se recibe como un argumento de línea de comandos para el generador de trabajo, de esta forma, solo es necesario modificar dicho parámetro en el archivo config.xml cuando hace la llamada al generador de trabajo.

```
<daemons>
      ...
      <daemon>
            <cmd>euler_work_generator --neuler 100 -d 3 </cmd>
      </daemon>
      ...
</daemons>
```

Esta característica se implementa en la función main del generador de trabajo, junto con las demás opciones de línea de comandos que se pueden emplear al ejecutar el generador, además se agregó la documentación de esta característica a la función usage(), que se encarga de proveer ayuda respecto a las características del generador. El siguiente fragmento de código muestra las implementación de dicha característica.

```
      ...
      } else if (!strcmp(argv[i], "--neuler")) {
            N = std::stoi(argv[++i],nullptr);
            exit(0);
      } else {
      ...
```

En la variable main_loop, se realiza la llamada a la función make_job, y se generan los valores de k para los archivos de trabajo, esta implementación se muestra en el siguiente fragmento de código:

```
      int njobs = (CUSHION-n)/REPLICATION_FACTOR;
```

```
            log_messages.printf(MSG_DEBUG,
                "Making %d jobs\n", njobs
            );
            for (int i=0; i<njobs; i++) {
                k++;
                if(k<=N){
                retval = make_job(k);
                if (retval) {
                          log_messages.printf(MSG_CRITICAL,"can't
make job: %s\n", boincerror(retval));
                          exit(retval);
                }
                }else{
                     log_messages.printf(MSG_CRITICAL,"Ya se
crearon %lu tareas, durmiendo generador de trabajo\n", ne);
                     daemon_sleep(50);
                }
```

El generador de trabajo generará N unidades de trabajo, dependiendo del valor que se le proporcione por línea de comandos.

5.2.5 VALIDADOR

Cuando se reciban los archivos de salida, será necesario validar que los resultados obtenidos sean correctos, cada plataforma maneja de forma diferente las variables de punto flotante, por lo que no hay forma de que el resultado proveniente de un cliente sea exactamente igual al de otro. Sin embargo, lo que si se sabe es que los resultados obtenidos no pueden tener el valor de cero y no puede ser negativo. Por tanto, el siguiente código se encarga de validar lo antes mencionado en la función init_result():

```
retval = try_fopen(fi.path.c_str(), f, "r");
    if (retval) return retval;
    if(!mpfr_inp_str(kterm, f, 10, MPFR_RNDD)){
     mpfr_clear(kterm);
     fclose(f);
     log_messages.printf(MSG_CRITICAL,
            "[RESULT#%lu %s] check_set: Imposible leer el tÃ©rmino
del archivo.\n",
            result.id, result.name
        );
     return ERR_FOPEN;
    }
    fclose(f);
```

```
    if(mpfr_sgn(kterm) <= 0){
     mpfr_clear(kterm);
     log_messages.printf(MSG_CRITICAL,
           "[RESULT#%lu %s] check_set: TÃ©rmino invÃ¡lido (menor
o igual a cero)\n",
           result.id, result.name
        );
     return ERR_EULER_NZERO;
    }
    mpfr_clear(kterm);
    return 0;
```

5.2.6 ASIMILADOR

Si los archivos recibidos contienen información valida, se mueven a un directorio especial, y se obtienen los valores para generar la constante de Euler. Esto se realiza a través del asimilador, empleando el código fuente provisto por BOINC, solo se modificó la función assimilate_handler(), se añade un apuntador a archivo, para leer la información de los archivos de salida, y se agrega el siguiente código para obtener los datos:

```
   f = boinc_fopen("e", "rb");
    if(f){
        mpfr_inp_str(e, f, 10, 8192, MPFR_RNDD);
        fclose(f);
    }
    else{
        mpfr_set_d(e, 0.0, MPFR_RNDD);
    }

    f = boinc_fopen(archivo, "rb");

    if(f){
        mpfr_inp_str(coef, f, 10, 8192, MPFR_RNDD);
        fclose(f);

        mpfr_add(e, e, coef, MPFR_RNDD);

    f = boinc_fopen("e", "wb");

    if(f) mpfr_out_str(f, 10, 8192, e, MPFR_RNDD);

    else write_error("Error al abrir el archivo");

    fclose(f);

    mpfr_clears(e, archivo);
```

Los tres programas DAEMON fueron modificados para utilizar la librería MPFR, por lo que es necesario modificar el makefile que se emplea para compilar dichos programas. Para este fin se agregan los enlaces a las líbrerias MPFR y GMP en la variable SERVERLIBS, que queda como sigue:

```
SERVERLIBS = $(LIBSCHED) $(LIBBOINC_CRYPT) $(LIBBOINC)
$(MYSQL_LIBS) $(PTHREAD_LIBS) $(RSA_LIBS) $(SSL_LIBS) -lmpfr -lgmp
```

Una vez que tenemos los códigos para la aplicación de trabajo y para los programas daemon del servidor se crea el proyecto como se muestra en la figura 5.22.

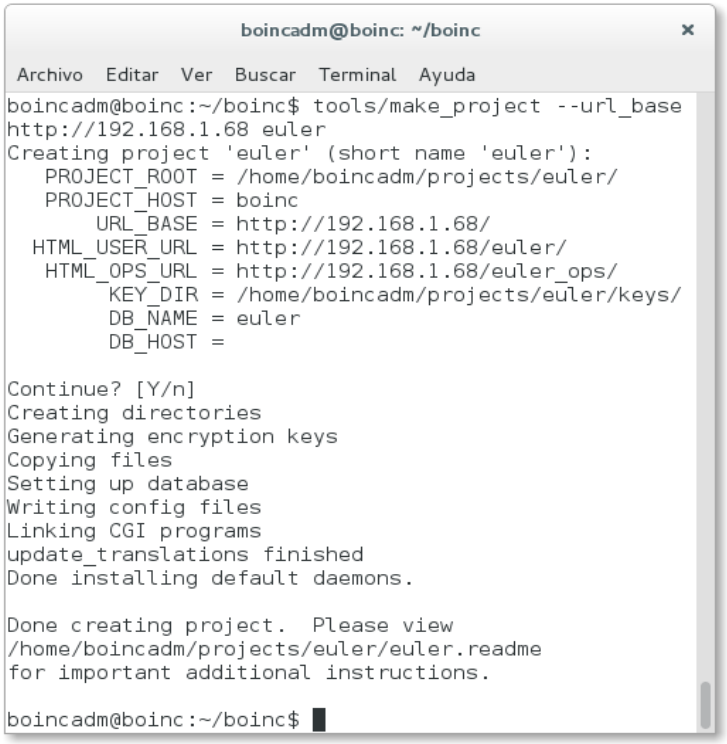


Figura 5.22 Creacion del proyecto euler con make_project

Se crean los directorios para la aplicación de trabajo en la carpeta apps, como se muestra en la imagen 5.23:

Figura 5.23 Creación de los directorios de la aplicación

Se copia la aplicación de trabajo a los directorios creados como se ilustra en la imagen 5.24
:

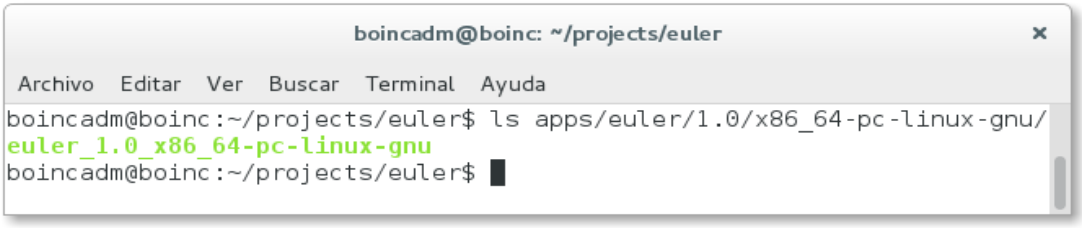


Figura 5.24 Aplicaciones de trabajo

Se crea el archivo que describe la versión de la aplicación, el cual se muestra en la figura 5.25.

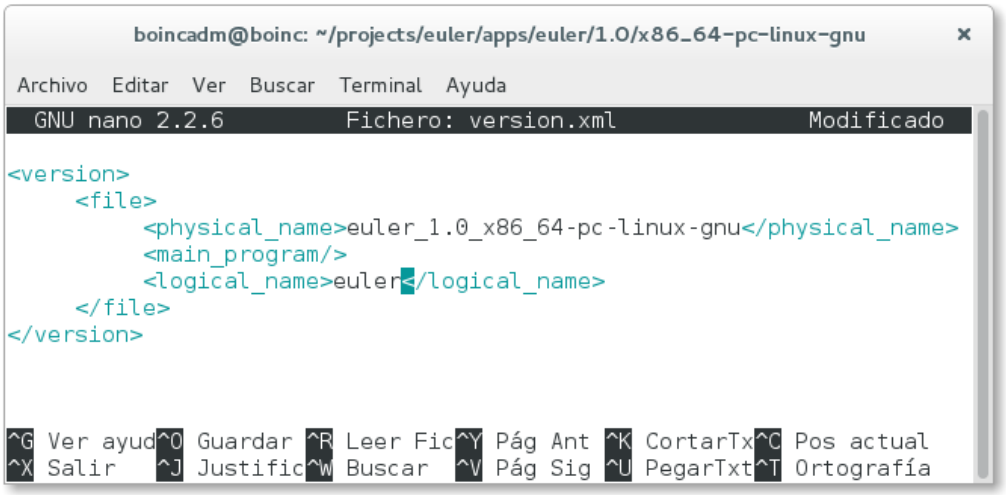


Figura 5.25 Archivo de descripción de la aplicación

Se modifica el archivo project.xml, para cargar las versiones de la aplicación para las plataformas objetivo, en este caso, la aplicación está dirigida a clientes con sistema operativo GNU/Linux de 64 bits para procesadores Intel y AMD, el contenido del archivo project.xml se muestra en la figura 5.26.

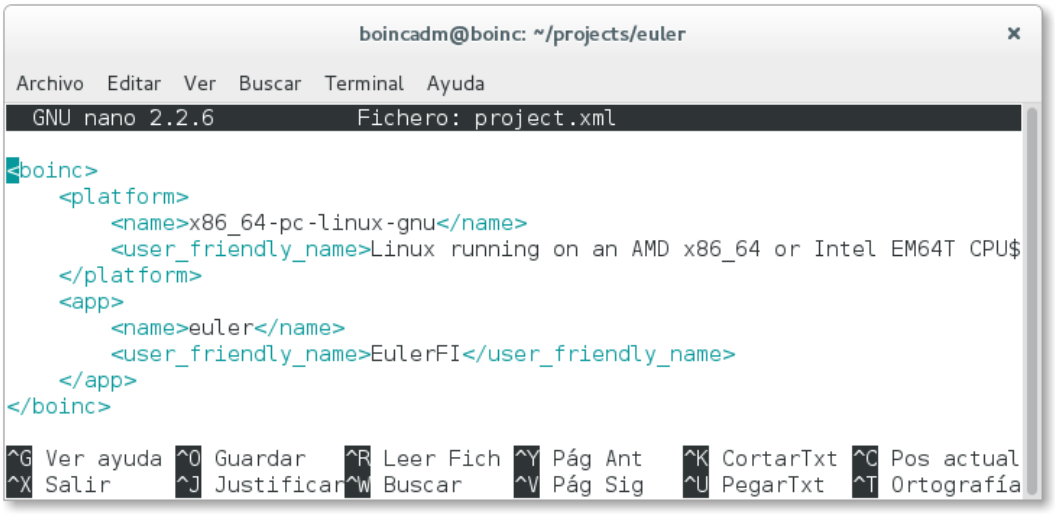


Figura 5.26 Archivo project.xml

Una vez modificado este archivo, se ejecuta el script xadd, la salida se ilustra en la figura 5.27

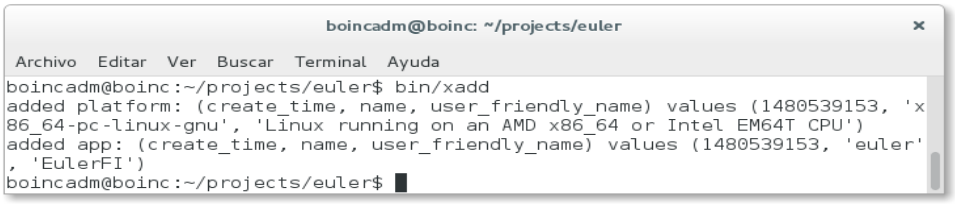


Figura 5.27 Ejecución del script xadd

Y se ejecuta el script update_versions, la salida se muestra en la figura 2.28

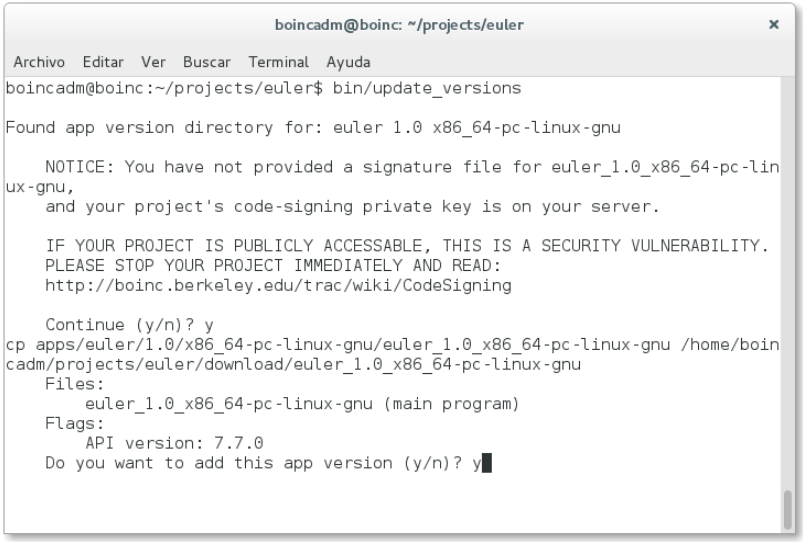


Figura 5.28 Ejecución del script update_versions

Se configura el proyecto en Apache, en la figura 5.29 se muestra la configuración del proyecto en el archivo apache.conf.

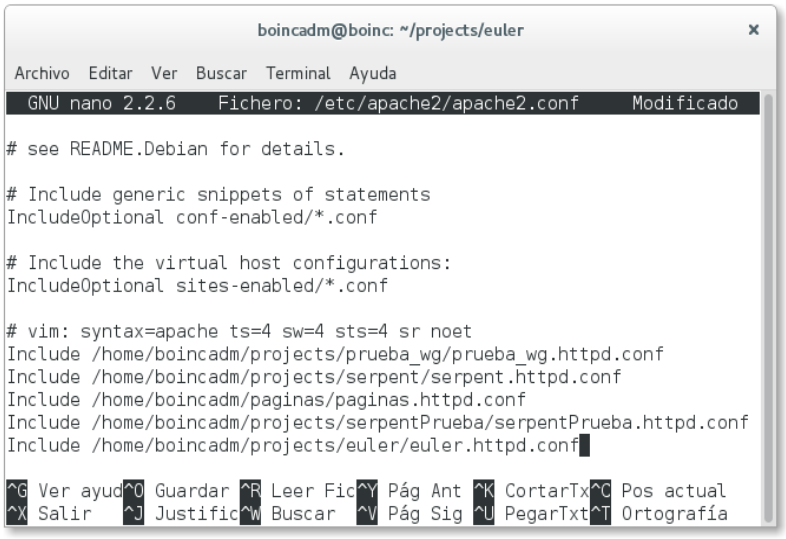


Figura 5.29 Archivo de configuración de Apache

Y en crontab se agrega la tarea como se muestra en la figura 5.30:

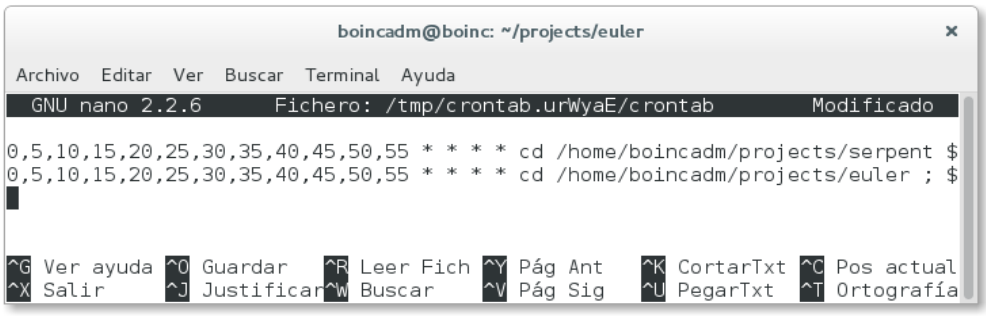


Figura 5.30 Archivo crontab

Se inicia el proyecto con el comando start, como se muestra en la figura 5.31.

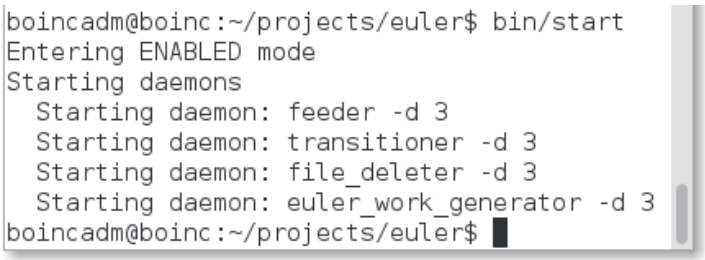


Figura 5.31 Comando start

Una vez realizada esta configuración, es posible acceder al sitio del proyecto como se muestra en la figura 5.32:

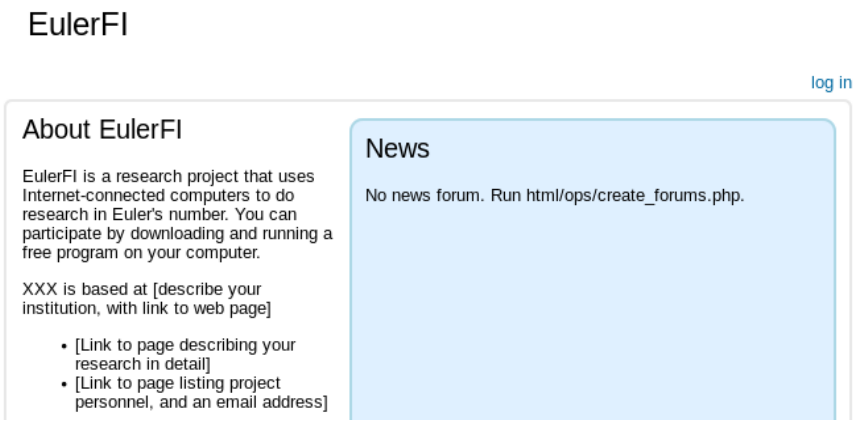


Figura 5.32 Página web de inicio del proyecto

Se crea una cuenta para conectarse al proyecto, a través de la página web como se muestra en la figura 5.33.

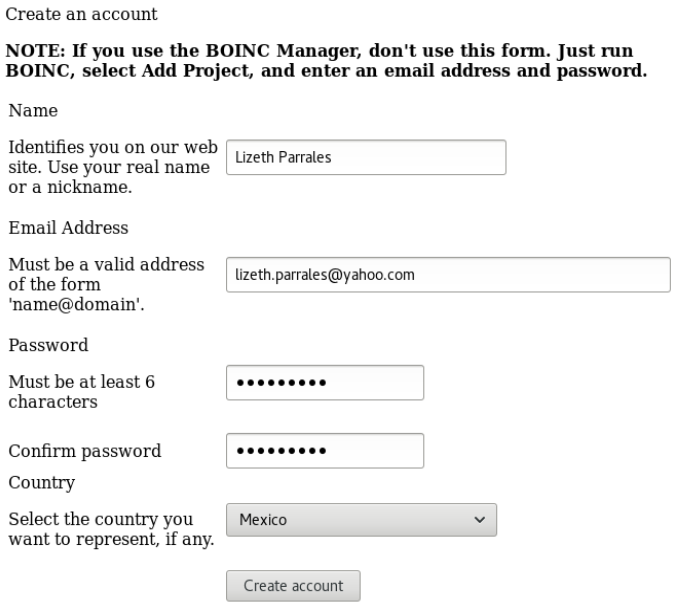


Figura 5.33 Creación de una cuenta desde la página web del proyecto

En este caso se realizó la conexión con el proyecto por línea de comandos como se muestra en la figura 5.34

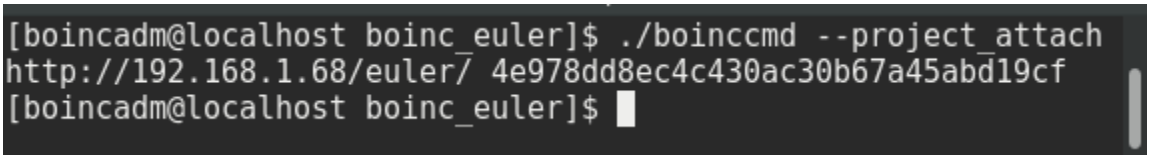


Figura 5.34 Conexión desde la computadora cliente

La computadora cliente en este caso, cuenta con un procesador core i3, y 4GB de memoria RAM, y ejecuta un sistema GNU/Linux Fedora 26 de 64 bits. La computadora servidor cuenta con un procesador Core 2 Duo y 2 GB de RAM y ejecuta el sistema operativo GNU/Linux Fedora 23 de 64 bits.

En otra terminal en la que previamente se ejecutó una instancia del núcleo del cliente, se puede ver el progreso de la ejecución de las tareas en la figura 5.35, como puede observarse, la ejecución de cada unidad de trabajo no tarda mas de 1 segundo, el proyecto esta configurado para generar 100 unidades de trabajo, aunque esto puede modificarse desde el script config.xml para aumentar el número de unidades de trabajo a enviar.

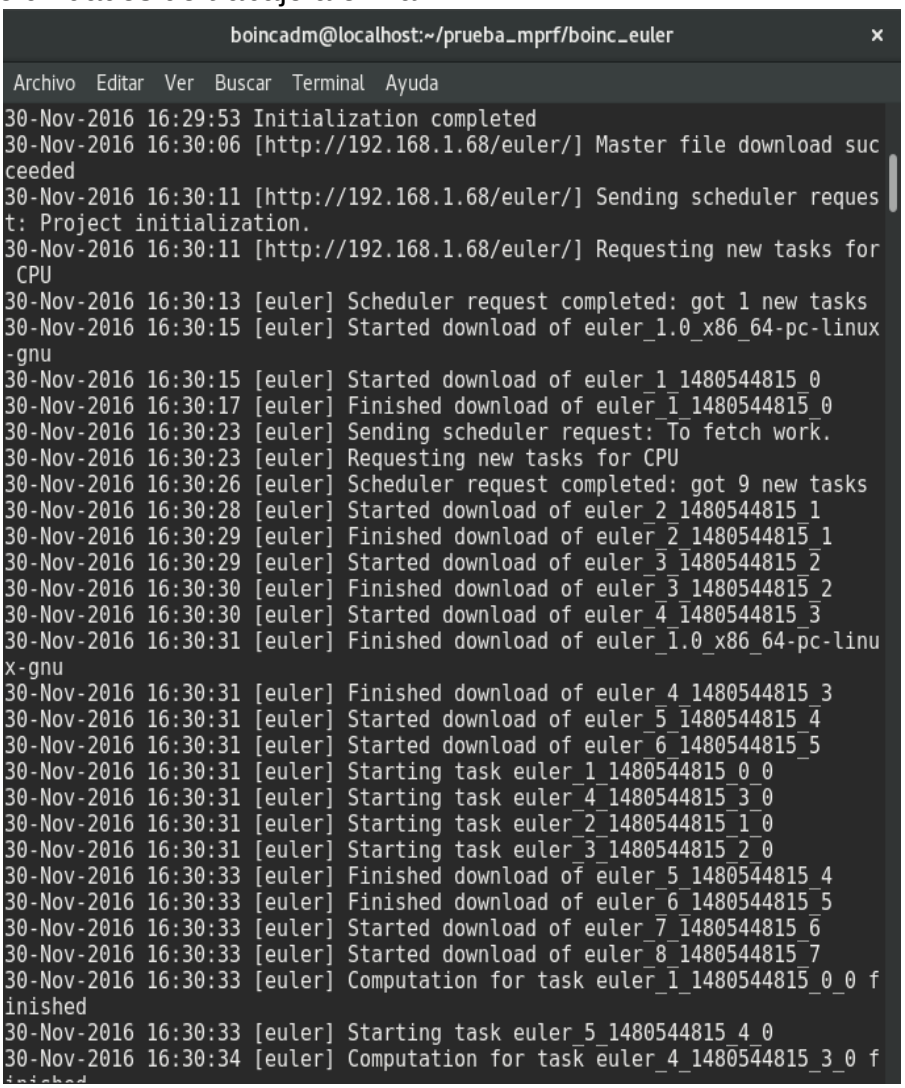


Figura 5.35 Visualización de las tareas desde la computadora cliente

RESULTADOS Y APORTACIONES

Como resultado del trabajo realizado, se obtuvo un servidor BOINC listo para soportar varios proyectos, tambien se realizaron varios ejemplos que permiten probar la eficiencia y capacidad de BOINC para proyectos cientificos, lo que proveera a la facultad una plataforma de computo distribuido que podrá ser empleada en proyectos de investigación que requieren grandes cantidades de recursos y que actualmente solo pueden ser cubiertas a través del uso de una supercomputadora, de la cual solo pueden disponer por un periodo limitado de tiempo.

El disponer de un servidor BOINC propio proveera a la facultad de ingeniería una cantidad de recursos comparable con la cantidad de recursos que provee una de las supercomputadoras mas robustas a nivel mundial, además, permitirá que un proyecto obtenga los recursos necesarios para su funcionamiento sin limitantes de tiempo, ya que un servidor puede alojar varios proyectos y un cliente puede trabajar en mas de un proyecto a la vez.

Aunado a esto, la investigación realizada servira para sentar las bases practicas para la materia Sistemas Distribuidos, lo que permitirá que los ingenieros de futuras generaciones obtengan las habilidades necesarias competir en un mercado poco explotado como lo es el computo distribuido y que gracias a la evolución en el desarrollo de hardware en la epoca actual se ha convertido en una tendencia que pronto se impondrá en el mercado mundial.

CONCLUSIONES

El computo paralelo ha permitido que se reduzcan los tiempos de ejecución de procesos que, de ser programados secuencialmente, tardarían miles de años en arrojar resultados útiles; el computo en paralelo ha contribuido a la resolución de problemas en distintas áreas del desarrollo humano, por ejemplo, el estudio del clima en nuestro planeta, investigación que se ha llevado a cabo desde la década de 1960 en el Laboratorio Lawrence Livermore. Otro ejemplo es el estudio de los fenómenos económicos en la sociedad moderna. El computo paralelo ha permitido que fenómenos físicos como el descubrimiento de nuevos tratamientos para enfermedades para las que aún no se encuentra una cura, o la simulación de las reacciones nucleares dentro de un reactor al emplear diversos materiales para su construcción, puedan ser analizadas sin necesidad de poner en riesgo la integridad de las personas involucradas en estas áreas, y además permite que fenómenos imposibles de reproducir en un laboratorio como la explosión de una supernova o la reproducción del clima del planeta en épocas anteriores a la existencia del hombre puedan ser reproducidas a través de simulaciones.

Aún que existen diversas implementaciones de cómputo paralelo, tanto a nivel de hardware como a nivel de software, el cómputo distribuido ha permitido que los tiempos de ejecución de los algoritmos paralelos sean mucho menores a los que la computación paralela convencional había permitido.

A diferencia de un sistema de cómputo paralelo que puede estar compuesto por cientos de equipos, un sistema distribuido puede llegar a estar compuesto por millones de dispositivos electrónicos, distribuidos geográficamente en cualquier lugar del planeta, además dichos elementos pueden tener distintas combinaciones de arquitecturas de hardware y software, lo que permite que el poder de procesamiento aportado por los elementos de estos sistemas llegue a igualar al poder de procesamiento de las supercomputadoras más robustas.

Debido a la gran cantidad de dispositivos conectados a la red hoy en día, los sistemas distribuidos son parte de nuestra vida cotidiana, en la forma de servicios tales como los buscadores web, o los servicios de la banca electrónica. Además, los sistemas de cómputo distribuido voluntario permiten que los investigadores obtengan los recursos de cómputo necesarios para el desarrollo de proyectos de investigación. BOINC a diferencia de otras plataformas de computo distribuido se presenta como una solución completa, robusta y flexible, desarrollada en lenguaje C/C++, lo que lo convierte en un software que puede adaptarse a las necesidades del proyecto. Además, provee soporte para dispositivos tales como GPUs y

smarphones, debido a las características de estos dispositivos es posible crear sistemas con una gran capacidad de poder de procesamiento.

GLOSARIO

Algoritmo: Conjunto de reglas que permiten resolver un problema con un número determinado de operaciones o pasos. Es una secuencia ordenada de pasos precisos, simples, no ambiguos y finitos.

Android: Sistema operativo movil, basado el kernel de GNU/Linux y desarrollado actualmente por Google. Con una interfaz de usuario basada en manipulación directa, ya que esta dirigido a dispositivos con pantallas tactiles tales como smartphones y tablets. El codigo fuente de android es lanzado por Google bajo una licencia de código abierto.

Apache: El proyecto Apache HTTP server es un esfuerzo por desarrollar y mantener un servidor HTTP de códogo abierto para sistemas operativos modernos tales como UNIX y Windows. La meta del proyecto es proveer un servidor seguro, eficiente y extensible que proporcione servicios basados en los estandares HTTP actuales.

API: (Application Programming Interface) Es un lenguaje y formato de mensajes utilizado por una aplicación para comunicarse con el sistema operativo u otro programa de control tal como un sistema manejador de base de datos o un protocolo de comunicación.

Applet: Es un programa pequeño, tal como un programa de utilidad o un procesador de archivos con funciones limitadas. Así se denomina a los programas en Java que se ejecutan desde un navegador web.

ARPANET: (Advanced Research Projects Agency NETwork) Es la red de investigación fundada por la Agencia de Investigación de Proyectos Avanzados (ARPA, por sus siglas en inglés). El software fue desarrollado por Bolt, Baranek y Newman (BBN), y se emplearan minicomputadoras Honeywell 516 como conmutadores de paquetes. ARPAnet fue lanzada 1969 en cuatro lugares incluyendo dos en los campus de la Universidad de California, el Instituto de Investigación de Stanford y la Universidad de Utah.

Arquitectura de computadora: La arquitectura de una computadora se enfoca en la forma en que se construyen las unidades logicas (cpu, unidad de memoria, unidad de e/s).

C: Lenguaje de programacion desarrollado por Dennis Ritchie el los Laboratorios Bell a principios de los 70s. Fue diseñado originalmente para elaborar sistemas operativos y compiladores. El lenguaje C combina las ventajas de un lenguaje de

alto nivel con las del lenguaje maquina. Debido a que es muy cercano al lenguaje maquina, es muy compacto y rapido. se le cpnsidera como un lenguaje de propisito general. Esta directamente relacionado con el sistema operativo UNIX debido a que el 93% de su codigo esta escrito en C y el 7% restante esta escrito en ensamblador. El lenguaje no está ligado a ninguna arquitectura de computadora en particular, a diferencia del lenguaje ensamblador.

C++: Fue desarrollado por Bjarne Stroustrup, cuya intención era implementar el paradigma orientado a objetos al lenguaje C, inicialmente el nombre del lenguaje era C con clases y el 1983 se renombró como C++.

CDC: (Control Data Corporation) fue fundada en Minneapolis en 1957 por un grupo de ingenieros, muchos de ellos provenientes de UNIVAC. Eventualmente se volvieron famosos como proveedores de sistemas de computo cientifico muy grandes y poderosos, muchos de los cuales fueron diseñados por Seymour Cray y su equipo hasta 1972 que salió de la compañía.

CGI (php): Es un estandar de programación web cuyo objetivo es ejecutar un programa en el servidor y desplegar el resultado al cliente. Surgió a inicios de los 90, se estandarizó rapidamente al permitir la creación de páginas web dinamicas.

Cinta magnetica: Cinta de material flexible recubierta, que almacena información en una de sus caras. Puede almacenar millones de caracteres, la longitud máxima estandar es de 2400 pies y un ancho de media pulgada.

Circuito integrado: Es un cristal semiconductor de silicio, denomiando chip, que contiene los componentes electrónicos para construir compuertas digitales. Las diversas compuertas se interconectan dentro del chip para formar el circuito requerido. El circuito se monta en un recipiente de ceramica o plastico, y las conexiones se sueldan a terminales externas para formar el circuito integrado.

Cluster: Un clúster es un conjunto de computadoras conectadas a través de una red de datos y coordinadas a través de un software especial denominado middleware, que se comporta como un único recurso de cómputo.

Comando make: Es un programa que mantiene un registro de las dependencias entre archivos, y solo actualiza aquellos archivos que han cambiado desde la ultima actualización (compilación, enlazado o eliminación de archivos temporales).

Concurrencia: Cuando se tiene mas de una tarea en un procesador con un solo un nucleo y el scheduler del sistema operativo cambia de una tarea a otra dando la sensación de que las tareas se ejecutan de forma simultanea.

CORBA: (Common Object Request Broker Architecture) es un estándar definido por el grupo de gestión de objetos (OMG) que permite el funcionamiento conjunto de componentes de software escritos en distintos lenguajes informáticos y que se ejecutan en distintos sistemas. CORBA es un estándar para distribuir objetos a través de redes de modo que las operaciones en dichos objetos puedan llamarse de forma remota. CORBA no está asociado con ningún lenguaje de programación en particular y cualquier lenguaje que tenga un enlace CORBA puede utilizarse para llamar e implementar objetos CORBA.

Cray: Fue una compañía fundada por Seymour Cray en 1972 con el fin de diseñar y construir las supercomputadoras del mas alto rendimiento a nivel mundial.
Cron: cron es una utilidad que permite ejecutar tareas en segundo plano en intervalos de tiempo regulares.

DARPA: (Defense Advanced Research Projects Agency) The U.S. military agency responsible for technology projects. Founded as the Advanced Research Projects Agency (ARPA) in 1958, it was renamed DARPA in 1972, then back to ARPA in 1993 and once again back to DARPA in 1996. DARPA is a small agency that assumes the role of technological engine for the Department of Defense and is involved in evaluating future systems. Over the years, DARPA has made major contributions to information technology.

Debian: El proyecto Debian es un grupo mundial de voluntarios que se esfuerzan por producir una distribución de sistema operativo que este compuesta enteramente de software libre. El producto principal del proyecto a la fecha es la distribución de software Debian GNU/Linux, la cual incluye a Linux como núcleo del sistema operativo, así como miles de aplicaciones pre-empaquetadas. Se soportan en mayor o menor medida distintos tipos de procesadores, incluyendo el procesador Intel i386 y superiores, y los procesadores Alpha, ARM, Intel IA-64, Motorola 68k, MIPS, PA-RISC, PowerPC, Sparc (y UltraSparc), IBM S/390 y Hitachi SuperH. Tambien existe una versión de Debian basada en el núcleo FreeBSD y se estan desarrollando versiones basadas en otros núcleos como el núcleo Hurd.

Discos magneticos: Es un medio de almacenamiento secundario de acceso aleatorio en formato de instalacion intercambiable o permanente.
Elementos de procesamiento: Son procesadores interconectados entre si de tal forma que pueden coordinarse para procesar una tarea especifica.
Fedora: El proyecto Fedora es una sociedad de desarrollo de software libre con miembros distribuidos alrededor de todo el mundo que produce el sistema operativo Fedora, el cual esta basado en el núcleo GNU/Linux.

Firewall: Es un dispositivo de seguridad que monitorea el trafico de una red (entrante o saliente) y decide si bloquea o permite el trafico de acuerdo a un conjunto especifico de reglas de seguridad.

FLOP: (FLoating point Operations Per Second) The measurement of floating point calculations. For example, 100 megaFLOPS (MFLOPS) is 100 million floating point operations per second, and 100 teraFLOPS (TFLOPS) is 100 trillion FLOPS.

FORTRAN: (FORmula TRANslator) Es un lenguaje de programación de alto nivel diseñado primordialmente para aplicaciones cientificas.

GIT: Es un proyecto de codigo abierto desarrollado por Linus Torvalds en 2005, cuyo objetivo es el desarrollo y mantenimiento de un sistema de control de versiones de software.

GNU/Linux: Es un sistema operativo que evolucionó del nucleo Linux, desarrollado por Linus Torvalds, empleando programas desarrollados por el proyecto GNU liderado por Richard Stallman.

Google: Es una compañia fundada por Sergey Brin y Larry Page cuyo primer producto desarrollado fue un motor de busqueda que se encarga de ordenar los resultados debido a su popularidad. La palabra Google proviene de un juego de palabras con el termino googol que se refiere a la representacion de un numero formado por un 1 y 100 ceros.

GPU: Es un componente de procesamiento de graficos dentro de un sistema de computo, creada con el fin de minimizar la carga de trabajo del procesador (CPU).

GROMACS: (Groningen MAchine for Chemical Simulation) that was developed at the University of Groningen, The Netherlands, in the early 1990s. The software, written in ANSI C, originates from a parallel hardware project, and is well suited for parallelization on processor clusters. By careful optimization of neighbor searching and of inner loop performance, GROMACS is a very fast program for molecular dynamics simulation.
Hardware: Se denomina hardware a los componentes físicos que componen un sistema de cómputo.

Intel: Es una compañia que desarrolla circuitos semiconductores para computadoras fundada en 1968 en Estados Unidos por Robert Noyce y Gordon Moore.

Java: Es un lenguaje de programación de proposito general, basado en clases y orientado a objetos; normalmente se compila a bytecode, el cual es un formato binario definido en las especificaciones de la maquina virtual de Java.

Laboratorio Lawrence Livermore: Laboratorio de seguridad nacional del gobierno de Estados Unidos, que se dedica a la investigación en areas como el clima del planeta y los campos de aplicación de la la energía nuclear ya sea para su uso como suministro de energía de Estados Unidos o para el desarrollo de armamento.

Mac: Sistema de computo personal desarrollado por Apple Computer.

Makefile: Es un archivo empleado por el comando make el cual contiene un conjunto de reglas que el comando make lee para generar una aplicación.

Máquina virtual: Es una emulación de un sistema de cómputo dentro de otro sistema a través de un software especializado para este fin.

Memoria compartida: En un entorno de procesamiento concurrente cada proceso tiene su propio segmento de memoria de trabajo, la memoria compartida, es un segmento de memoria al cual pueden acceder todos estos procesos.

Memoria de núcleo de ferrita: Fue un tipo de memoria que estaba construida por anillos de un material denominado ferrita, el cual es una estructura cristalina de hierro altamente magnetizable, el cúal permitía almacenar información por grandes periodos de tiempo, las memorias de ferrita estaban constituidas por matrices de anillos de ferrita. Fueron empleadas durante el periodo de 1953 a 1970 cuando fueron reemplazadas por las memorias semiconductoras.

Memoria distribuida: En un sistema distribuido, cada procesador tiene su propia memoria asociada, y solo tiene acceso a dicha memoria.

Método de Monte Carlo: Es un metodo estadistico que permite obtener soluciones aproximadas a problemas matemáticos o físicos haciendo factible la realización de experimentos con muestreos de números pseudoaleatorios de forma computacional.

Middleware: Es software que se encuentra entre las aplicaciones y el sistema operativo, las pilas de protocolos de red y el hardware. Sus principales funciones son: servir de puente entre las aplicaciones y el hardware de bajo nivel y la infraestructura de software con el fin de coordinar como se conectan e interoperan algunas partes de la aplicación; y tambien tiene como función habilitar y simplificar la integración de componentes desarrollados por diferentes proveedores.

MIMD: (Multiple Instruction Multiple Data stream) en esta organización multiples elementos de procesamiento se encuentran bajo el control de multiples unidades de control, cada unidad de control maneja un flujo de instrucciones las cuales son procesadas por el elemento de procesamiento correspondiente, y ademas a cada uno le corresponde su propio flujo de datos proveniente de la memoria principal.

MISD: (Multiple Instruction Single Data stream) en esta organización multiples elementos de procesamiento se encuentran bajo el control de multiples unidades de control, cada unidad de control maneja un flujo de intrucciones las cuales son procesadas por el elemento de procesamiento correspondiente, sin embargo, cada elemento de procesamiento trabaja sobre un solo flujo de datos en un ciclo de reloj. Esta organización no es muy utilizada en computadoras comerciales.

Modelo cliente/servidor: Dentro del modelo cliente/servidor, un cliente es un programa que utiliza servicios que otros programas que los proveen, denominados servidores. El cliente realiza una petición por un servicio, y el servidor ejecuta ese servicio. Las funciones que realiza el servidor normalmente requieren del manejo de recursos, por lo que el servidor sincroniza y administra el acceso a los recursos y responde la peticion del cliente con datos o con el status de la información.

Modelo de cadenas Markov: Las cadenas de Markov son un tipo de proceso estocastico que se emplea para describir procesos discretos que evolucionan en el tiempo (generaciones) o en el espacio (secuencias biológicas). El modelo de cadenas de Marcov se emplea para el modelado de procesos biológicos.

Modelo de paso de mensajes: En un sistema distribuido, cada procesador tiene su propia memoria asociada. Por lo que una aplicación se ejecuta como una colección de procesos autonomos, y cada proceso se comunica con otro enviando y recibiendo mensajes, cuando se envían datos en un mensaje, estos se envían desde la memoria local donde se ejecuta un proceso hasta la memoria local del proceso que recibe el mensaje.

MySQL: Es un manejador de base de datos de código abierto desarrollada y distribuida por Oracle Corporation.

NASA: (National Aeronautics and Space Administration) Es una organización creada en Octubre de 1958 en Estados Unidos, que se encarga de investigaciones en aeronautica, ciencias del espacio y sus aplicaciones.

Peer-to-Peer: es una arquitectura de comunicación que permite la comunicación entre los nodos que constituyen la red de forma directa sin necesidad de la información pase por un servidor.

PHP: (PHP es un acrónimo recursivo de PHP: Hypertext Preprocessor): Es un lenguaje de programación de código abierto para desarrollo web que puede incrustarse en HTML. El código que se escribe en PHP se ejecuta del lado del servidor, el cual genera código HTML que envia al cliente.

Plataforma (BOINC): Una plataforma es un objetivo de compilación, es decir, la combinación de un sistema operativo y una arquitectura de procesador especifica. Una versión de aplicación siempre esta asociada con una plataforma.

Playstation3: es una consola de videojuegos desarrollada por Sony Interactive Entertainment, que consta de un procesador IBM Cell BE que es capaz de procesar 153 GFLOPS en precisión simple, tiene una memoria principal de 256 MB y ademas cada elemento de procesamiento sinergico esta provisto de una memoria de 256 KB. Gracias a estas caracteristicas, puede ser utilizado para crear clusteres que trabajen en aplicaciones cientificas.

POO: Es un paradigma de programación en el cual la estructura del software esta basada en objetos que interactuan entre si para realizar una tarea especifica. Los objetos se comunican estre si por medio del envío y recepción de mensajes, como respuesta a un mensaje un objeto puede realizar una acción.

Procesador: Dentro de un sistema de cómputo, es la unidad encargada de ejecutar las instrucciones y procesar los datos necesarios para ejecutar tareas especificas. Consta de unan unidad aritmetica-logica y una unidad de control.
Procesador vectorial: Es un tipo de procesador especializado que puede decodificar instrucciones cuyos operandor son vectores.

Protocolo de comunicación: Son reglas establecidas para regular la forma en que los datos se transmiten en una red de computadoras.

Proyecto SERENDIP: Es un proyecto de la universidad de Berkeley en California que se encarga de buscar señales de radio emitidas por civilizaciones extraterrestres, SERENDIP es un acronimo para Search for Extraterrestrial Radio Emissions from Nearby Developed Intelligent Populations.
Punto flotante: es una forma de representación de números en notación cientifica dentro de una computadora. Esta forma de representación descompone el número a representar en dos partes, una mantisa la cual contiene los dígitos del número y un exponente que indica en donde se colocará el punto decimal en relación al inicio de la mantisa.

Radiotelescopio: Un radiotelescopio es un dispositivo que se encarga de captar ondas de radio provenientes del espacio, se utiliza debido a que algunos astros no

emiten luz visible, pero es posible observarlos gracias a las ondas de radio que emiten.

Semiconductor: Sustancia cristalina, utilizada en la fabricación de circuitos integrados debido a sus caracteristicas de conductividad eléctrica.

Servidor LAMP: Se denomina así a un conjunto de software libre que se emplea para habilitar un servidor de contenido web, el termino LAMP es un acronimo para el software que lo compone: el sistema operativo Linux, el servidor Apache, el manejador de bases de datos MySQL y PHP.

SIMD: (Single Instruction Multiple Data streams) En esta organización multiples elementos de procesamiento trabajan dirigidos por una sola unidad de control ejecutan una misma instrucción enviada desde la unidad de control, sin embargo, cada elemento de procesamiento ejecuta dicha instrucción sobre su propio flujo de datos, proveniente de su propia memoria.

SISD: (Single Instruction and Single Data stream) De acuerdo a la clasificacion de Flynn, en esta organización un solo elemento de procesamiento contenido en un unico CPU realiza la ejecución de instrucciones secuenciales.

Sistema distribuido: Es un sistema formado por un conjunto de nodos computacionales autónomos coordinados y conectados a través de una red que permite la transferencia de datos con el propósito de realizar una tarea específica común o compartir recursos.

Sistema operativo: Es una capa de software que se encarga de controlar los recursos de un sistema de cómputo y ademas provee una base para el desarrollo y la ejecución de aplicaciones.

Socket: Es una combinación de IP y número de puerto en un enlace de comunicación establecido entre dos programas que se ejecutan en una red de datos, el número de puerto permite que la capa TCP identifique la aplicación hacia la que van dirigidos los datos.
Software: Es un conjunto de instrucciones que procesa una computadora. Existen dos tipos de software; el software de sistema se encarga de administrar los recursos de hardware y de la ejecución de aplicaciones, y el software de aplicación se encarga de procesar datos para el usuario.

SSL: Es un protocolo de seguridad de internet, desarrollado por Netscape, se encarga de validar la identidad de un sitio web, de cifrar la información de transacciones y de asegurar la transmision de datos sin errores.

Subrutina: Es una abstracción de una llamada a un proceso, el programa que realiza la llamada le envia a la subrutina parametros que la subrutina toma como argumentos, la subrutina ejecuta sus intrucciones, las cuales pueden causar efectos globales en el estado del sistema y en algunos casos regresa los resultados al programa que realizo el llamado.

Supercomputadora: Son computadoras en las cuales se emplean varios procesadores que trabajan de forma coordinada, pueden ser computadoras de tipo SIMD o MIMD.

Tablas hash: Son una implementación de arreglos asociativos o diccionarios. Un arreglo asociativo puede tener un conjunto de índices muy grande, potencialmente infinito, del cual solo un pequeño numero es utilizado.

Tarjetas perforadas: Fue una de las primeras formas de introducción de información empleado en maquinas automaticas y en computadoras de primera generación.

Taxonomía de Flynn: Clasificación de las computadoras paralelas propuesta por Michael Flynn en 1972, la cual no se basa en la arquitectura de las computadoras, sino en los conceptos de flujos de instrucciones y de flujos de datos.

TCP: (Protocolo de control de transmision) es un protocolo confiable orientado a la conexion que permite que un flujo de bytes que se origina en una maquina se entregue sin errores a cualquier otra maquina de la interred.

TENEX: Fue un sistema operativo muy utilizado en las computadoras DEC-10.
Transistor: Componente electrónico de estado solido formado por uniones semiconductores de tipo PNP ó NPN. el tipo de transistor implica una polarizacion determinada.

Tubos de vacío: Es un dispositivo electronico inventado por John Ambrose Fleming en 1904, el cual permitía rectificar una señal de corriente alterna a corriente directa, posteriormente Lee de Forrest lo mejoró agregando un electrodo mas creando un triodo. La primera generación de computadoras empleaba componentes hechos con tubos de vacio y aun se emplean para algunos dispositivos de telecomunicaciones.

UDP: (User Datagram Protocol) Proporciona una forma para que las aplicaciones envien datagramas IP encapsulados sin tener que establecer una conexión.
UNIX: Se nombra así a los sistemas operativos derivados del sistema operativo AT&T UNIX desarrollado por Ken Thompson, Dennis Ritchie y otros en los laboratorios Bell en 1970s.

Versión (BOINC): Para BOINC, una versión de aplicación, se refiere a una aplicación de trabajo compilada para una plataforma (sistema operativo y arquitectura de procesador) especifica.

VLSI: Se refiere a la tecnología de fabricación de circuitos integrados en la que pueden implementarse desde 100,000 hasta 1,000,000 de transistores en un chip.

Windows: Sistema operativo desarrollado por la empresa Microsoft en 1985, cuyo desarrollo fue liderado por Bill Gates, fundador de la compañia, la interfaz gráfica de la primera versión de Windows se ejecutaba sobre el sistema MS-DOS.

Workstation: El termino hace referencia a una computadora de alto rendimiento orientada a usuarios profesionales, tales como desarrolladores de software o diseñadores graficos, mas que a consumidores comunes, emplean CPUs mas veloces y poseen una capacidad de memoria RAM mayor a comparación de un equipo comercial. Tambien se emplea este termino para denominar a cualquier computadora cliente dentro de una red de datos.

XEROX: Es una compañia fundada en 1906 en Rochester con el nombre de The Haloid Photografic Company posteriormente cambio a Haloid Xerox y finalmente a XEROX, inicialmente fue una compañia dedicada a la fabricación de papel y equipo fotografico, sin embargo, en su centro de investigación en Palo Alto, desarrolló la primera computadora con interfaz gráfica de usuario en 1973.

ANEXO A: DIAGRAMAS DE FLUJO DEL SERVIDOR Y EL CLIENTE DE BOINC

DIAGRAMA DE FLUJO DEL SERVIDOR DE BOINC

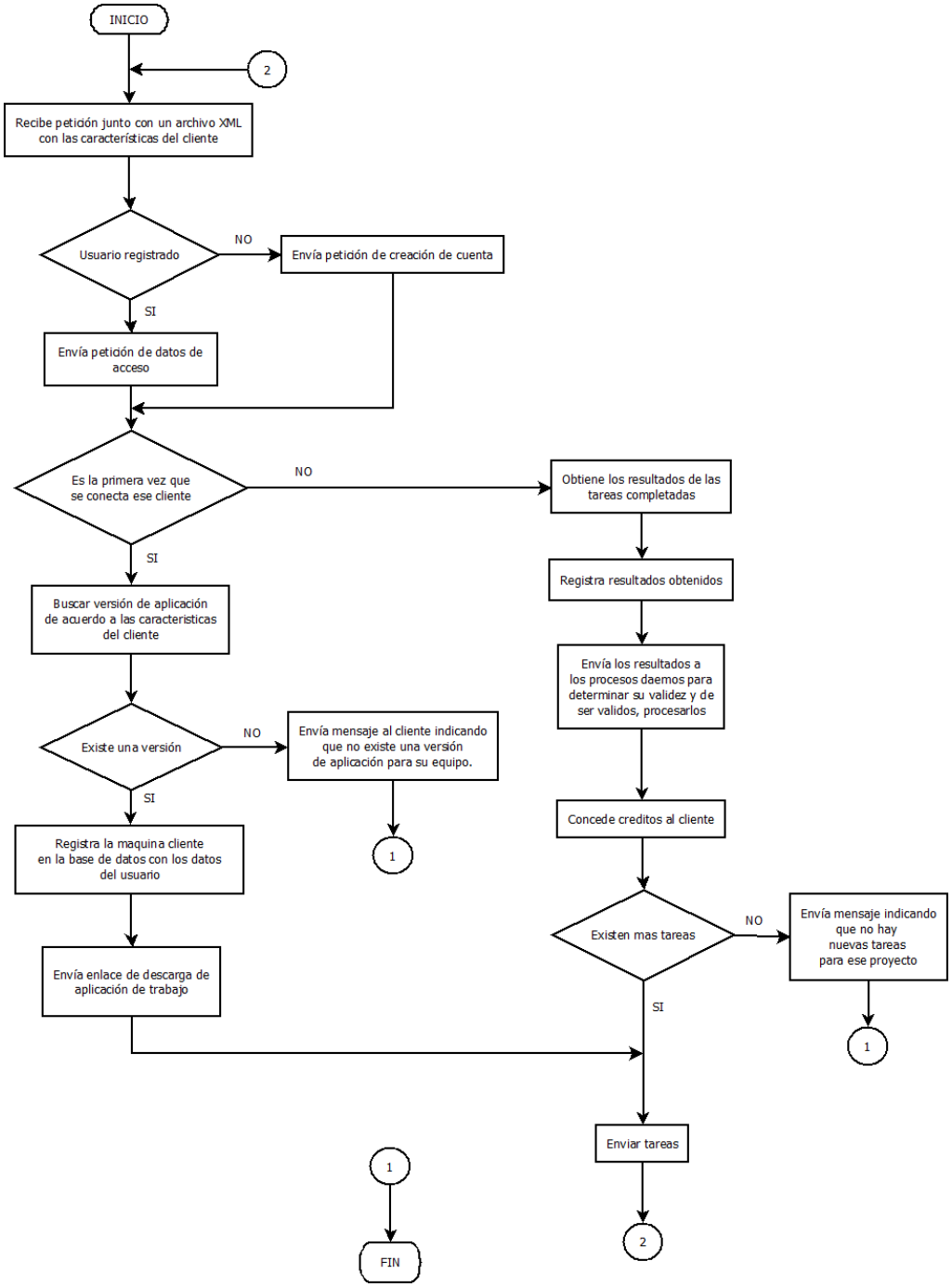
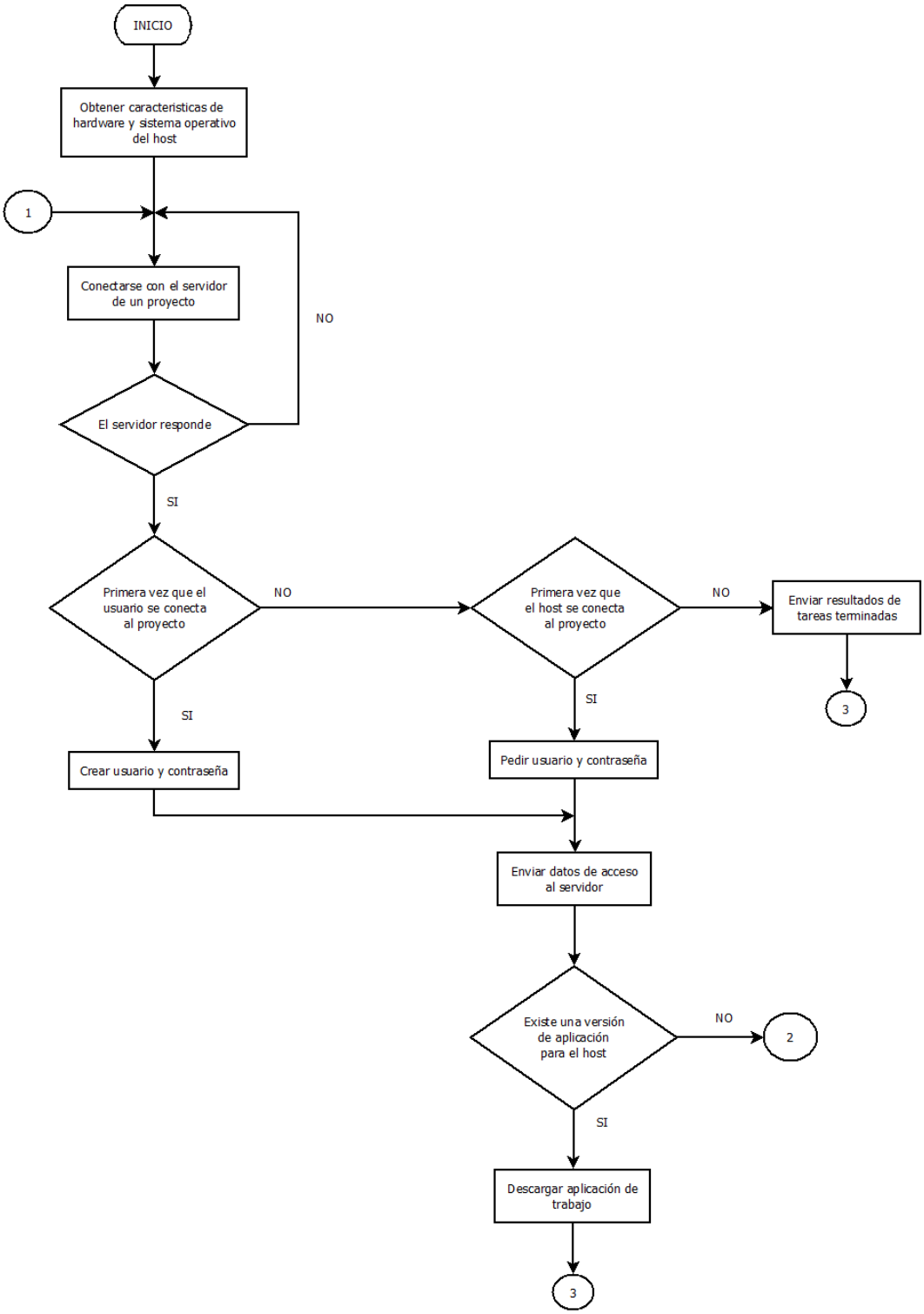
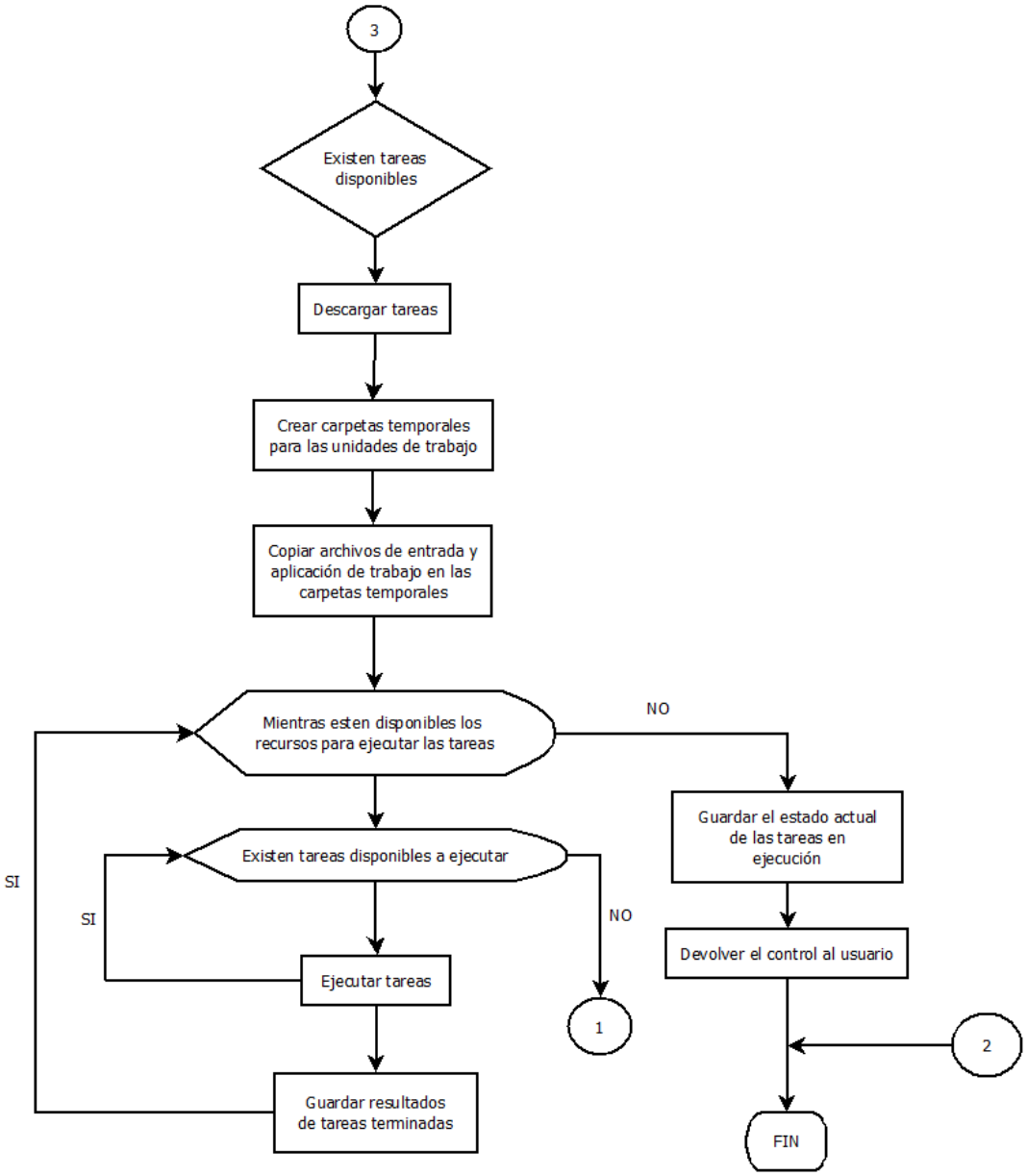


DIAGRAMA DE FLUJO DEL CLIENTE DE BOINC




3
Existen tareas disponibles
Descargar tareas
Crear carpetas temporales para las unidades de trabajo
Copiar archivos de entrada y aplicación de trabajo en las carpetas temporales
Mientras esten disponibles los recursos para ejecutar las tareas
NO
Guardar el estado actual de las tareas en ejecución
Devolver el control al usuario
2
FIN
Existen tareas disponibles a ejecutar
SI
NO
1
SI
Ejecutar tareas
Guardar resultados de tareas terminadas

ANEXO B: SALIDA DE LOS SCRIPTS DE CONFIGURACIÓN DE BOINC

SALIDA DEL COMANDO ./CONFIGURE (COMPILACIÓN DE BOINC)

```
[lizeth@localhost boinc]$ ./configure --disable-server --
disable-client --disable-manager
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking target system type... x86_64-unknown-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... /usr/bin/gawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking whether make supports nested variables... (cached)
yes
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc3
checking for g++... /usr/bin/g++
checking whether we are using the GNU C++ compiler... yes
checking whether /usr/bin/g++ accepts -g... yes
checking dependency style of /usr/bin/g++... gcc3
checking for g++... /usr/bin/g++
checking whether we are using the GNU Objective C++
compiler... no
checking whether /usr/bin/g++ accepts -g... no
checking dependency style of /usr/bin/g++... gcc3
checking for g77... no
```

```
checking for xlf... no
checking for f77... no
checking for frt... no
checking for pgf77... no
checking for cf77... no
checking for fort77... no
checking for fl32... no
checking for af77... no
checking for xlf90... no
checking for f90... no
checking for pgf90... no
checking for pghpf... no
checking for epcf90... no
checking for gfortran... no
checking for g95... no
checking for xlf95... no
checking for f95... no
checking for fort... no
checking for ifort... no
checking for ifc... no
checking for efc... no
checking for pgfortran... no
checking for pgf95... no
checking for lf95... no
checking for ftn... no
checking for nagfor... no
checking whether we are using the GNU Fortran 77 compiler...
no
checking whether accepts -g... no
checking how to run the C preprocessor... gcc -E
checking whether make sets $(MAKE)... (cached) yes
checking for ln... /usr/bin/ln
checking whether '/usr/bin/ln' works... yes
checking whether ln -s works... yes
checking whether 'ln -s' really works or whether I'm deluding
myself... it works
checking whether gcc and cc understand -c and -o together...
yes
checking if C compiler supports -Wall... yes
checking if C++ compiler supports -Wall... yes
checking if f77 compiler supports -Wall... no
--- Configuring BOINC 7.7.0 (Release) ---
```

```
--- Build Components: ( libraries) ---
checking for docbook2x-man... no
checking for grep that handles long lines and -e...
/usr/bin/grep
checking for egrep... /usr/bin/grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking whether we are compiling for cygwin... no
checking for winsock2.h... (cached) no
checking for winsock.h... (cached) no
checking windows.h usability... no
checking windows.h presence... no
checking for windows.h... no
checking sys/socket.h usability... yes
checking sys/socket.h presence... yes
checking for sys/socket.h... yes
checking dependency style of gcc... gcc3
checking how to print strings... printf
checking for a sed that does not truncate output...
/usr/bin/sed
checking for fgrep... /usr/bin/grep -F
checking for ld used by gcc... /usr/bin/ld
checking if the linker (/usr/bin/ld) is GNU ld... yes
checking for BSD- or MS-compatible name lister (nm)...
/usr/bin/nm -B
checking the name lister (/usr/bin/nm -B) interface... BSD nm
checking the maximum length of command line arguments...
1572864
checking whether the shell understands some XSI constructs...
yes
checking whether the shell understands "+="... yes
checking how to convert x86_64-unknown-linux-gnu file names
to x86_64-unknown-linux-gnu format... func_convert_file_noop
```

```
checking how to convert x86_64-unknown-linux-gnu file names
to toolchain format... func_convert_file_noop
checking for /usr/bin/ld option to reload object files... -r
checking for objdump... objdump
checking how to recognize dependent libraries... pass_all
checking for dlltool... dlltool
checking how to associate runtime and link libraries...
printf %s\n
checking for ar... /usr/bin/ar
checking for archiver @FILE support... @
checking for strip... strip
checking for ranlib... ranlib
checking command to parse /usr/bin/nm -B output from gcc
object... ok
checking for sysroot... no
checking for mt... no
checking if : is a manifest tool... no
checking for dlfcn.h... yes
checking for objdir... .libs
checking if gcc supports -fno-rtti -fno-exceptions... no
checking for gcc option to produce PIC... -fPIC -DPIC
checking if gcc PIC flag -fPIC -DPIC works... yes
checking if gcc static flag -static works... no
checking if gcc supports -c -o file.o... yes
checking if gcc supports -c -o file.o... (cached) yes
checking whether the gcc linker (/usr/bin/ld -m elf_x86_64)
supports shared libraries... yes
checking whether -lc should be explicitly linked in... no
checking dynamic linker characteristics... GNU/Linux ld.so
checking how to hardcode library paths into programs...
immediate
checking for shl_load... no
checking for shl_load in -ldld... no
checking for dlopen... no
checking for dlopen in -ldl... yes
checking whether a program can dlopen itself... yes
checking whether a statically linked program can dlopen
itself... yes
checking whether stripping libraries is possible... yes
checking if libtool supports shared libraries... yes
checking whether to build shared libraries... yes
checking whether to build static libraries... yes
```

```
checking how to run the C++ preprocessor... /usr/bin/g++ -E
checking for ld used by /usr/bin/g++... /usr/bin/ld -m
elf_x86_64
checking if the linker (/usr/bin/ld -m elf_x86_64) is GNU
ld... yes
checking whether the /usr/bin/g++ linker (/usr/bin/ld -m
elf_x86_64) supports shared libraries... yes
checking for /usr/bin/g++ option to produce PIC... -fPIC -
DPIC
checking if /usr/bin/g++ PIC flag -fPIC -DPIC works... yes
checking if /usr/bin/g++ static flag -static works... no
checking if /usr/bin/g++ supports -c -o file.o... yes
checking if /usr/bin/g++ supports -c -o file.o... (cached)
yes
checking whether the /usr/bin/g++ linker (/usr/bin/ld -m
elf_x86_64) supports shared libraries... yes
checking dynamic linker characteristics... (cached) GNU/Linux
ld.so
checking how to hardcode library paths into programs...
immediate
checking default bitness of compiler... 64
checking boinc platform... x86_64-pc-linux-gnu
checking alternate boinc platform... i686-pc-linux-gnu
checking library extension... a
checking shared object extension... so
checking for shmget in dynamic library cygipc... no
checking for aio_fork in dynamic library aio... no
checking for dlopen in dynamic library dl...  -ldl
checking for gethostbyname in static library nsl...  -lnsl
checking for static library freetype... no
checking for dynamic library freetype... no
checking for bind in static library socket... no
checking for bind in dynamic library socket... no
checking for gzopen in static library z... no
checking for gzopen in dynamic library z... no
checking for md5_finish in dynamic library cups... no
checking for the pthreads library -lpthreads... no
checking whether pthreads work without any flags... no
checking whether pthreads work with -Kthread... no
checking whether pthreads work with -kthread... no
checking for the pthreads library -llthread... no
checking whether pthreads work with -pthread... yes
```

```
checking for joinable pthread attribute...
PTHREAD_CREATE_JOINABLE
checking if more special flags are required for pthreads...
no
checking for PTHREAD_PRIO_INHERIT... yes
checking for X... libraries , headers
checking for gethostbyname... yes
checking for connect... yes
checking for remove... yes
checking for shmat... yes
checking for IceConnectionNumber in -lICE... yes
checking for pthread_join in LIBS= with CFLAGS=-pthread...
yes
checking for joinable pthread attribute...
PTHREAD_CREATE_JOINABLE
checking if more special flags are required for pthreads...
no
checking for PTHREAD_PRIO_INHERIT... (cached) yes
checking whether we are using the Microsoft C compiler... no
checking for windows.h... (cached) no
checking for OpenGL library... -L/usr/local/lib -lGL
checking for OpenGL Utility library... -L/usr/local/lib -lGLU
checking for fopen in -lXmu... yes
checking for fopen in -lXi... yes
checking for GLUT library... -L/usr/local/lib -lglut
checking gl.h usability... no
checking gl.h presence... no
checking for gl.h... no
checking glu.h usability... no
checking glu.h presence... no
checking for glu.h... no
checking glut.h usability... no
checking glut.h presence... no
checking for glut.h... no
checking glaux.h usability... no
checking glaux.h presence... no
checking for glaux.h... no
checking GL/gl.h usability... yes
checking GL/gl.h presence... yes
checking for GL/gl.h... yes
checking GL/glu.h usability... yes
checking GL/glu.h presence... yes
```

```
checking for GL/glu.h... yes
checking GL/glut.h usability... yes
checking GL/glut.h presence... yes
checking for GL/glut.h... yes
checking GL/glaux.h usability... no
checking GL/glaux.h presence... no
checking for GL/glaux.h... no
checking OpenGL/gl.h usability... no
checking OpenGL/gl.h presence... no
checking for OpenGL/gl.h... no
checking OpenGL/glu.h usability... no
checking OpenGL/glu.h presence... no
checking for OpenGL/glu.h... no
checking OpenGL/glut.h usability... no
checking OpenGL/glut.h presence... no
checking for OpenGL/glut.h... no
checking OpenGL/glaux.h usability... no
checking OpenGL/glaux.h presence... no
checking for OpenGL/glaux.h... no
checking GLUT/glut.h usability... no
checking GLUT/glut.h presence... no
checking for GLUT/glut.h... no
checking MesaGL/gl.h usability... no
checking MesaGL/gl.h presence... no
checking for MesaGL/gl.h... no
checking MesaGL/glu.h usability... no
checking MesaGL/glu.h presence... no
checking for MesaGL/glu.h... no
checking MesaGL/glut.h usability... no
checking MesaGL/glut.h presence... no
checking for MesaGL/glut.h... no
checking MesaGL/glaux.h usability... no
checking MesaGL/glaux.h presence... no
checking for MesaGL/glaux.h... no
checking libnotify/notify.h usability... no
checking libnotify/notify.h presence... no
checking for libnotify/notify.h... no
checking gtk/gtk.h usability... no
checking gtk/gtk.h presence... no
checking for gtk/gtk.h... no
checking locale.h usability... yes
checking locale.h presence... yes
```

```
checking for locale.h... yes
checking xlocale.h usability... yes
checking xlocale.h presence... yes
checking for xlocale.h... yes
checking for jpeg_start_compress in -ljpeg... yes
checking jpeglib.h usability... yes
checking jpeglib.h presence... yes
checking for jpeglib.h... yes
checking for dirent.h that defines DIR... yes
checking for library containing opendir... none required
checking for ANSI C header files... (cached) yes
checking for sys/wait.h that is POSIX.1 compatible... yes
checking whether time.h and sys/time.h may both be
included... yes
checking return type of signal handlers... void
checking for sys/types.h... (cached) yes
checking sys/un.h usability... yes
checking sys/un.h presence... yes
checking for sys/un.h... yes
checking arpa/inet.h usability... yes
checking arpa/inet.h presence... yes
checking for arpa/inet.h... yes
checking dirent.h usability... yes
checking dirent.h presence... yes
checking for dirent.h... yes
checking grp.h usability... yes
checking grp.h presence... yes
checking for grp.h... yes
checking fcntl.h usability... yes
checking fcntl.h presence... yes
checking for fcntl.h... yes
checking for inttypes.h... (cached) yes
checking for stdint.h... (cached) yes
checking for memory.h... (cached) yes
checking netdb.h usability... yes
checking netdb.h presence... yes
checking for netdb.h... yes
checking netinet/in.h usability... yes
checking netinet/in.h presence... yes
checking for netinet/in.h... yes
checking netinet/tcp.h usability... yes
checking netinet/tcp.h presence... yes
```

```
checking for netinet/tcp.h... yes
checking netinet/ether.h usability... yes
checking netinet/ether.h presence... yes
checking for netinet/ether.h... yes
checking net/if.h usability... yes
checking net/if.h presence... yes
checking for net/if.h... yes
checking net/if_arp.h usability... yes
checking net/if_arp.h presence... yes
checking for net/if_arp.h... yes
checking signal.h usability... yes
checking signal.h presence... yes
checking for signal.h... yes
checking for strings.h... (cached) yes
checking sys/auxv.h usability... yes
checking sys/auxv.h presence... yes
checking for sys/auxv.h... yes
checking sys/file.h usability... yes
checking sys/file.h presence... yes
checking for sys/file.h... yes
checking sys/fcntl.h usability... yes
checking sys/fcntl.h presence... yes
checking for sys/fcntl.h... yes
checking sys/ipc.h usability... yes
checking sys/ipc.h presence... yes
checking for sys/ipc.h... yes
checking sys/ioctl.h usability... yes
checking sys/ioctl.h presence... yes
checking for sys/ioctl.h... yes
checking sys/msg.h usability... yes
checking sys/msg.h presence... yes
checking for sys/msg.h... yes
checking sys/param.h usability... yes
checking sys/param.h presence... yes
checking for sys/param.h... yes
checking sys/resource.h usability... yes
checking sys/resource.h presence... yes
checking for sys/resource.h... yes
checking sys/select.h usability... yes
checking sys/select.h presence... yes
checking for sys/select.h... yes
checking sys/sem.h usability... yes
```

```
checking sys/sem.h presence... yes
checking for sys/sem.h... yes
checking sys/shm.h usability... yes
checking sys/shm.h presence... yes
checking for sys/shm.h... yes
checking sys/sockio.h usability... no
checking sys/sockio.h presence... no
checking for sys/sockio.h... no
checking for sys/socket.h... (cached) yes
checking for sys/stat.h... (cached) yes
checking sys/statvfs.h usability... yes
checking sys/statvfs.h presence... yes
checking for sys/statvfs.h... yes
checking sys/statfs.h usability... yes
checking sys/statfs.h presence... yes
checking for sys/statfs.h... yes
checking sys/systeminfo.h usability... no
checking sys/systeminfo.h presence... no
checking for sys/systeminfo.h... no
checking sys/time.h usability... yes
checking sys/time.h presence... yes
checking for sys/time.h... yes
checking for sys/types.h... (cached) yes
checking sys/utsname.h usability... yes
checking sys/utsname.h presence... yes
checking for sys/utsname.h... yes
checking sys/vmmeter.h usability... no
checking sys/vmmeter.h presence... no
checking for sys/vmmeter.h... no
checking for sys/wait.h... (cached) yes
checking for unistd.h... (cached) yes
checking utmp.h usability... yes
checking utmp.h presence... yes
checking for utmp.h... yes
checking errno.h usability... yes
checking errno.h presence... yes
checking for errno.h... yes
checking procfs.h usability... no
checking procfs.h presence... no
checking for procfs.h... no
checking ieeefp.h usability... no
checking ieeefp.h presence... no
```

```
checking for ieeefp.h... no
checking setjmp.h usability... yes
checking setjmp.h presence... yes
checking for setjmp.h... yes
checking float.h usability... yes
checking float.h presence... yes
checking for float.h... yes
checking sal.h usability... no
checking sal.h presence... no
checking for sal.h... no
checking intrin.h usability... no
checking intrin.h presence... no
checking for intrin.h... no
checking x86intrin.h usability... yes
checking x86intrin.h presence... yes
checking for x86intrin.h... yes
checking pmmintrin.h usability... yes
checking pmmintrin.h presence... yes
checking for pmmintrin.h... yes
checking xmmintrin.h usability... yes
checking xmmintrin.h presence... yes
checking for xmmintrin.h... yes
checking emmintrin.h usability... yes
checking emmintrin.h presence... yes
checking for emmintrin.h... yes
checking immintrin.h usability... yes
checking immintrin.h presence... yes
checking for immintrin.h... yes
checking avxintrin.h usability... no
checking avxintrin.h presence... no
checking for avxintrin.h... no
checking whether _xgetbv is declared... no
checking whether xgetbv is declared... no
checking whether __xgetbv is declared... no
checking whether cpuid is declared... no
checking whether _cpuid is declared... no
checking whether __cpuid is declared... no
checking if assembler supports xgetbv... yes
checking for nvapi.h... no
checking for socklen_t... yes
checking for net/if.h... (cached) yes
checking for net/if_arp.h... (cached) yes
```

```
checking for sys/sysctl.h... yes
checking for sys/mount.h... yes
checking for sys/swap.h... yes
checking for sys/sensors.h... no
checking for resolv.h... yes
checking for netinet/if_ether.h... yes
checking for struct lifconf... no
checking for struct lifreq... no
checking for struct ifconf... yes
checking for struct ifreq... yes
checking for struct ether_addr... yes
checking for special C compiler options needed for large
files... no
checking for _FILE_OFFSET_BITS value needed for large
files... no
checking whether largefile support breaks C++... no
checking standard C++ headers... yes
checking for C++ header <algorithm>... (cached) yes
checking for C++ header <bitset>... (cached) yes
checking for C++ header <cassert>... (cached) yes
checking for C++ header <cctype>... (cached) yes
checking for C++ header <cerrno>... (cached) yes
checking for C++ header <cfloat>... (cached) yes
checking for C++ header <climits>... (cached) yes
checking for C++ header <clocale>... (cached) yes
checking for C++ header <cmath>... (cached) yes
checking for C++ header <complex>... (cached) yes
checking for C++ header <csetjmp>... (cached) yes
checking for C++ header <csignal>... (cached) yes
checking for C++ header <cstdarg>... (cached) yes
checking for C++ header <cstddef>... (cached) yes
checking for C++ header <cstdio>... (cached) yes
checking for C++ header <cstdlib>... (cached) yes
checking for C++ header <cstring>... (cached) yes
checking for C++ header <ctime>... (cached) yes
checking for C++ header <deque>... (cached) yes
checking for C++ header <fstream>... (cached) yes
checking for C++ header <functional>... (cached) yes
checking for C++ header <iomanip>... (cached) yes
checking for C++ header <ios>... (cached) yes
checking for C++ header <iosfwd>... (cached) yes
checking for C++ header <iostream>... (cached) yes
```

```
checking for C++ header <istream>... (cached) yes
checking for C++ header <iterator>... (cached) yes
checking for C++ header <limits>... (cached) yes
checking for C++ header <list>... (cached) yes
checking for C++ header <locale>... (cached) yes
checking for C++ header <map>... (cached) yes
checking for C++ header <memory>... (cached) yes
checking for C++ header <numeric>... (cached) yes
checking for C++ header <ostream>... (cached) yes
checking for C++ header <queue>... (cached) yes
checking for C++ header <set>... (cached) yes
checking for C++ header <sstream>... (cached) yes
checking for C++ header <stack>... (cached) yes
checking for C++ header <stdexcept>... (cached) yes
checking for C++ header <streambuf>... (cached) yes
checking for C++ header <string>... (cached) yes
checking for C++ header <utility>... (cached) yes
checking for C++ header <valarray>... (cached) yes
checking for C++ header <vector>... (cached) yes
checking for C++ namespaces... yes
checking for min(0,0) in namespace std... yes
checking for max(0,0) in namespace std... yes
checking for transform((char *)0,(char *) 0,(char
*)0,(int(*)(int))malloc) in namespace std... yes
checking for locale("") in namespace std... yes
checking whether gcc needs -traditional... no
checking for vprintf... yes
checking for _doprnt... no
checking for ether_ntoa... yes
checking for setpriority... yes
checking for sched_setscheduler... yes
checking for strlcpy... no
checking for strlcat... no
checking for strcasestr... yes
checking for strcasecmp... yes
checking for sigaction... yes
checking for getutent... yes
checking for setutent... yes
checking for getisax... no
checking for strdup... yes
checking for _strdup... no
checking for strdupa... no
```

```
checking for _strdupa... no
checking for daemon... yes
checking for stat64... yes
checking for putenv... yes
checking for setenv... yes
checking for unsetenv... yes
checking for res_init... no
checking for strtoull... yes
checking for localtime... yes
checking for localtime_r... yes
checking for gmtime... yes
checking for gmtime_r... yes
checking whether _fpreset is declared... no
checking whether fpreset is declared... no
checking whether _configthreadlocale is declared... no
checking for an ANSI C-conforming const... yes
checking for size_t... yes
checking whether struct tm is in sys/time.h or time.h...
time.h
checking for struct tm.tm_zone... yes
checking for sin in dynamic library m...  -lm
checking for pthread_join in dynamic library pthread...  -
lpthread
checking for dynamic library nvapi... no
checking for res_init in -lresolv... no
checking for res_query in static library resolv... no
checking for res_query in dynamic library resolv... no
checking for whoami... /usr/bin/whoami
checking that generated files are newer than configure...
done
configure: creating ./config.status
config.status: creating version.h
config.status: creating api/Makefile
config.status: creating apps/Makefile
config.status: creating clientgui/Makefile
config.status: creating clientgui/res/Makefile
config.status: creating clientscr/Makefile
config.status: creating client/Makefile
config.status: creating client/win/boinc_path_config.py
config.status: creating client/scripts/Makefile
config.status: creating client/scripts/boinc-client
config.status: creating db/Makefile
```

```
config.status: creating doc/Makefile
config.status: creating doc/manpages/Makefile
config.status: creating html/Makefile
config.status: creating lib/Makefile
config.status: creating locale/Makefile
config.status: creating Makefile
config.status: creating py/Boinc/version.py
config.status: creating py/Makefile
config.status: creating py/boinc_path_config.py
config.status: creating py/setup.py
config.status: creating sched/boinc_path_config.py
config.status: creating sched/Makefile
config.status: creating packages/generic/sea/Makefile
config.status: creating packages/solaris/CSW/Makefile
config.status: creating
packages/solaris/CSW/boinccclient/Makefile
config.status: creating
packages/solaris/CSW/boinccclient/pkginfo
config.status: creating
packages/solaris/CSW/boinccclient/prototype
config.status: creating
packages/solaris/CSW/boincdevel/Makefile
config.status: creating
packages/solaris/CSW/boincdevel/pkginfo
config.status: creating
packages/solaris/CSW/boincdevel/prototype
config.status: creating
packages/solaris/CSW/boinclibs/Makefile
config.status: creating
packages/solaris/CSW/boinclibs/pkginfo
config.status: creating
packages/solaris/CSW/boinclibs/prototype
config.status: creating
packages/solaris/CSW/boincmanager/Makefile
config.status: creating
packages/solaris/CSW/boincmanager/pkginfo
config.status: creating
packages/solaris/CSW/boincmanager/prototype
config.status: creating samples/Makefile
config.status: creating test/boinc_path_config.py
config.status: creating test/Makefile
config.status: creating test/version.inc
```

```
config.status: creating tools/boinc_path_config.py
config.status: creating tools/Makefile
config.status: creating vda/Makefile
config.status: creating zip/Makefile
config.status: creating zip/zip/Makefile
config.status: creating zip/unzip/Makefile
config.status: creating m4/Makefile
config.status: creating config.h
config.status: executing depfiles commands
config.status: executing libtool commands
--- Configuring BOINC 7.7.0 (Release) ---
--- Build Components: ( libraries) ---
[lizeth@localhost boinc]$
```

SALIDA DEL COMANDO MAKE (COMPILACIÓN DE BOINC)

```
[lizeth@localhost boinc]$ make
cd . && sh generate_svn_version.sh
make  all-recursive
make[1]: se ingresa al directorio `/home/lizeth/boinc'
Making all in m4
make[2]: se ingresa al directorio `/home/lizeth/boinc/m4'
make[2]: No se hace nada para `all'.
make[2]: se sale del directorio `/home/lizeth/boinc/m4'
Making all in api
make[2]: se ingresa al directorio `/home/lizeth/boinc/api'
  CXX      boinc_api.lo
  CXX      graphics2_util.lo
  CXX      reduce_main.lo
  CXXLD    libboinc_api.la
  CXX      libboinc_graphics2_la-gutil.lo
  CXX      libboinc_graphics2_la-gutil_text.lo
  CXX      libboinc_graphics2_la-reduce_lib.lo
  CXX      libboinc_graphics2_la-graphics2.lo
  CXX      libboinc_graphics2_la-graphics2_unix.lo
  CXXLD    libboinc_graphics2.la
  CXX      boinc_opencl.lo
  CXXLD    libboinc_opencl.la
rm -f libboinc_api.a
/usr/bin/ln .libs/libboinc_api.a .
rm -f libboinc_graphics2.a
/usr/bin/ln .libs/libboinc_graphics2.a .
rm -f libboinc_opencl.a
/usr/bin/ln .libs/libboinc_opencl.a .
make[2]: se sale del directorio `/home/lizeth/boinc/api'
Making all in lib
make[2]: se ingresa al directorio `/home/lizeth/boinc/lib'
  CXX      libboinc_la-app_ipc.lo
  CXX      libboinc_la-base64.lo
  CXX      libboinc_la-cc_config.lo
  CXX      libboinc_la-cert_sig.lo
  CXX      libboinc_la-coproc.lo
  CXX      libboinc_la-diagnostics.lo
diagnostics.cpp:699:6: warning: unused parameter 'siginfo' [-
Wunused-parameter]
```

```
 void boinc_catch_signal(int signal, struct siginfo *siginfo,
void *sigcontext) {
      ^
diagnostics.cpp:699:6: warning: unused parameter 'sigcontext'
[-Wunused-parameter]
  CXX      libboinc_la-filesys.lo
  CXX      libboinc_la-gui_rpc_client.lo
  CXX      libboinc_la-gui_rpc_client_ops.lo
  CXX      libboinc_la-gui_rpc_client_print.lo
  CXX      libboinc_la-hostinfo.lo
  CXX      libboinc_la-md5.lo
  CXX      libboinc_la-md5_file.lo
  CXX      libboinc_la-mem_usage.lo
  CXX      libboinc_la-mfile.lo
  CXX      libboinc_la-miofile.lo
  CXX      libboinc_la-msg_log.lo
  CXX      libboinc_la-network.lo
  CXX      libboinc_la-notice.lo
  CXX      libboinc_la-opencl_boinc.lo
  CXX      libboinc_la-parse.lo
  CXX      libboinc_la-prefs.lo
  CXX      libboinc_la-procinfo.lo
  CXX      libboinc_la-proc_control.lo
  CXX      libboinc_la-proxy_info.lo
  CXX      libboinc_la-shmem.lo
  CXX      libboinc_la-str_util.lo
  CXX      libboinc_la-url.lo
  CXX      libboinc_la-util.lo
  CXX      libboinc_la-procinfo_unix.lo
  CXX      libboinc_la-synch.lo
  CXX      libboinc_la-unix_util.lo
  CXXLD    libboinc.la
rm -f libboinc.a
/usr/bin/ln .libs/libboinc.a .
make[2]: se sale del directorio `/home/lizeth/boinc/lib'
Making all in doc
make[2]: se ingresa al directorio `/home/lizeth/boinc/doc'
make[3]: se ingresa al directorio `/home/lizeth/boinc/doc'
make[3]: No se hace nada para `all-am'.
make[3]: se sale del directorio `/home/lizeth/boinc/doc'
make[2]: se sale del directorio `/home/lizeth/boinc/doc'
make[2]: se ingresa al directorio `/home/lizeth/boinc'
```

```
cd . && sh generate_svn_version.sh
make[2]: se sale del directorio `/home/lizeth/boinc'
make[1]: se sale del directorio `/home/lizeth/boinc'
[lizeth@localhost boinc]$
```

EJECUCIÓN DE XADD Y UPDATE_VERSIONS

```
boincadm@zeus:~/projects/appliz1$ bin/xadd
Processing <Platform#None windows_intelx86> ...
  Skipped existing <Platform#None windows_intelx86>
Processing <Platform#None windows_x86_64> ...
  Skipped existing <Platform#None windows_x86_64>
Processing <Platform#None i686-pc-linux-gnu> ...
  Skipped existing <Platform#None i686-pc-linux-gnu>
Processing <Platform#None x86_64-pc-linux-gnu> ...
  Skipped existing <Platform#None x86_64-pc-linux-gnu>
Processing <Platform#None powerpc-apple-darwin> ...
  Skipped existing <Platform#None powerpc-apple-darwin>
Processing <Platform#None i686-apple-darwin> ...
  Skipped existing <Platform#None i686-apple-darwin>
Processing <Platform#None x86_64-apple-darwin> ...
  Skipped existing <Platform#None x86_64-apple-darwin>
Processing <Platform#None sparc-sun-solaris2.7> ...
  Skipped existing <Platform#None sparc-sun-solaris2.7>
Processing <Platform#None sparc-sun-solaris> ...
  Skipped existing <Platform#None sparc-sun-solaris>
Processing <Platform#None sparc64-sun-solaris> ...
  Skipped existing <Platform#None sparc64-sun-solaris>
Processing <Platform#None powerpc64-ps3-linux-gnu> ...
  Skipped existing <Platform#None powerpc64-ps3-linux-gnu>
Processing <Platform#None arm-android-linux-gnu> ...
  Skipped existing <Platform#None arm-android-linux-gnu>
Processing <Platform#None anonymous> ...
  Skipped existing <Platform#None anonymous>
Processing <App#None example_app> ...
  Skipped existing <App#None example_app>
boincadm@zeus:~/projects/appliz1$ cat project.xml
<boinc>
    <platform>
        <name>sparc64-sun-linux-gnu</name>
        <user_friendly_name>Linux running on a SPARC 64-bit
CPU</user_friendly_name>
    </platform>
    <app>
        <name>appliz1</name>
        <user_friendly_name>Aplicacion de
Liz</user_friendly_name>
```

```
    </app>
</boinc>
You have new mail in /var/mail/boincadm
boincadm@zeus:~/projects/appliz1$ bin/xadd
Processing <Platform#None sparc64-sun-linux-gnu> ...
  Committed <Platform#53 sparc64-sun-linux-gnu> ; values:
{'_dirty': False,
 '_lazy_lookups': {},
 'create_time': 1433438827L,
 'deprecated': 0,
 'id': 53L,
 'name': 'sparc64-sun-linux-gnu',
 'user_friendly_name': 'Linux running on a SPARC 64-bit CPU'}
Processing <App#None appliz1> ...
  Committed <App#5 appliz1> ; values:
{'_dirty': False,
 '_lazy_lookups': {},
 'beta': 0,
 'create_time': 1433438827L,
 'deprecated': 0,
 'fraction_done_exact': 0,
 'homogeneous_app_version': 0,
 'homogeneous_redundancy': 0,
 'host_scale_check': 0,
 'id': 5L,
 'locality_scheduling': 0L,
 'min_avg_pfc': 1.0,
 'min_version': 0L,
 'n_size_classes': 0,
 'name': 'appliz1',
 'non_cpu_intensive': 0,
 'target_nresults': 0,
 'user_friendly_name': 'Aplicacion de Liz',
 'weight': 1.0}
boincadm@zeus:~/projects/appliz1$ bin/update_versions

Found app version directory for: appliz1 1.0 sparc64-sun-
linux-gnu

    NOTICE: You have not provided a signature file for
appliz1_1.0_sparc64-sun-linux-gnu,
```

```
    and your project's code-signing private key is on your
server.

    IF YOUR PROJECT IS PUBLICLY ACCESSABLE, THIS IS A
SECURITY VULNERABILITY.
    PLEASE STOP YOUR PROJECT IMMEDIATELY AND READ:
    http://boinc.berkeley.edu/trac/wiki/CodeSigning

    Continue (y/n)? y
cp apps/appliz1/1.0/sparc64-sun-linux-
gnu/appliz1_1.0_sparc64-sun-linux-gnu
/home/boincadm/projects/appliz1/download/appliz1_1.0_sparc64-
sun-linux-gnu
PHP Notice:  Undefined property: stdClass::$api_version in
/home/boincadm/projects/appliz1/bin/update_versions on line
451

Notice: Undefined property: stdClass::$api_version in
/home/boincadm/projects/appliz1/bin/update_versions on line
451
    Files:
        appliz1_1.0_sparc64-sun-linux-gnu (main program)
    Flags:
        API version: 7.5.0
    Do you want to add this app version (y/n)? y
    App version added successfully; ID=1
boincadm@zeus:~/projects/appliz1$
```

ANEXO C: CÓDIGOS FUENTE

MAKEFILE DE LA PRIMERA APLICACIÓN DE PRUEBA APPLIZ

```
BOINC_DIR = ../..
BOINC_API_DIR = $(BOINC_DIR)/api
BOINC_LIB_DIR = $(BOINC_DIR)/lib
CXXFLAGS += -g \
    -I$(BOINC_DIR) \
    -I$(BOINC_LIB_DIR) \
    -I$(BOINC_API_DIR) \
    -L.

PROGS = appliz

all: $(PROGS)

libstdc++.a:
     ln -s `g++ -print-file-name=libstdc++.a`

clean: distclean

distclean:
     /bin/rm -f $(PROGS) *.o libstdc++.a

appliz: appliz.o libstdc++.a $(BOINC_API_DIR)/libboinc_api.a
$(BOINC_LIB_DIR)/libboinc.a
     $(CXX) $(CXXFLAGS) $(CPPFLAGS) $(LDFLAGS) -o appliz
appliz.o libstdc++.a -pthread \
     $(BOINC_API_DIR)/libboinc_api.a \
     $(BOINC_LIB_DIR)/libboinc.a
```

CÓDIGO FUENTE DE LA APLICACIÓN APPLIZ

```
/****************************************
* Primer programa para BOINC que calcula  *
* la siguiente ecuación:                  *
*        r = x^2 + y                      *
* para una serie de valores de 'x' y 'y'  *
* almacenados en el archivo de entrada    *
* archivo appliz.cpp                      *
* Parrales Romay Guadalupe Lizeth         *
****************************************/

#include<stdio.h>
#include "boinc_api.h"
#include "filesys.h"
#include "mfile.h"

#define ENTRADA "in"
#define SALIDA "out"

using std::string;

void calcula(FILE *ap, MFILE r){
     float x,y,res = 0;

     while(!feof(ap)){
          fscanf(ap,"%f,%f\n",&x,&y);
          printf("x= %f, y=%f\n",x,y);
          res = x*x+y;
          r.printf("%f\n",res);
     }
     r.flush();
}
int main(){

     FILE *in;
     MFILE out;
     int retval;
```

```
    char input_path[512], output_path[512], chkpt_path[512];
    char buf[256];
    double fsize, fd;

    retval = boinc_init();

    if(retval){
        fprintf(stderr, "ERROR!!! %s boinc_init regreso
%d\n",
                boinc_msg_prefix(buf, sizeof(buf)),
retval);
        exit(retval);
    }

    fprintf(stderr, "App iniciada\n");

    boinc_resolve_filename(ENTRADA, input_path,
sizeof(input_path));

    fprintf(stderr,"El archivo de entrada esta en:
%s",input_path);

    in = boinc_fopen(input_path, "r");

    if(!in){
        fprintf(stderr,"ERROR!! %s BOINC no encontro el
archivo en
             %s",boinc_msg_prefix(buf, sizeof(buf)),
                input_path);
        exit(-1);
    }

    file_size(input_path, fsize);

    boinc_resolve_filename(SALIDA, output_path,
                           sizeof(output_path));
    retval = out.open(output_path,"wb");

    calcula(in, out);
```

```
    boinc_finish(0);
}
```

PRIMER GENERADOR DE TRABAJO DE EJEMPLO

```
/**************************************************************
********
Ejemplo                          obtenido                          de
http://boinc.berkeley.edu/trac/wiki/WorkGeneration
**************************************************************
*******/

#include "boinc_db.h"
#include "backend_lib.h"

int main() {
// Se utiliza un objeto de tipo DB_APP de la libreria
// boinc_db.h
    DB_APP app;
// Se crea una unidad de trabajo creando un objeto de tipo
//DB_WORKUNIT de la libreria boinc_db.h
    DB_WORKUNIT wu;
    char* wu_template;
    const char* infiles[] = {"infile"};
    char path[1024];

    SCHED_CONFIG config;
    config.parse_file();
    boinc_db.open(config.db_name, config.db_host,
    config.db_user, config.db_passwd);
    app.lookup("where name='myappname'");

    // write input file in the download directory
    // Se generan los archivos de entrada en el directorio de
//descarga
    config.download_path("infile", path);
    FILE* f = fopen(path, "w");
    fwrite(f, "random stuff");
    fclose(f);

    read_file_malloc("templates/input_template.xml",
     wu_template);

    wu.clear();     // guarda ceros en todos los campos

    //wu.name: Nombre de la unidad de trabajo, debe ser único
```

```
    //entre todas las unidades de trabajo del proyecto, para
esto
    //se agrega al nombre el PID del proceso que la creó, un
    //número de secuencia y un timestamp de UNIX.
strcpy(wu.name, "test_name");

//wu.appid: El identificador de la aplicación
    wu.appid = app.id;

    //wu.min_quorum: cantidad mínima de quórum, se ejecuta el
    //validador cuanto se alcanza esta cantidad mínima de
    //resultados exitosos, si la estricta mayoria concuerda
    //se consideran correctos
    wu.min_quorum = 2;

    //wu.target_nresults: Número de resultados que se crearán
    //inicialmente. Debe ser por lo menos igual a min_quorum,
    //o puede ser mayor, para reflejar el radio de pérdida de
    //resultados o para obtener el quórum más rápido
    wu.target_nresults = 2;

    //wu.max_error_results: Es el número máximo de resultados
    //con error permitidos, si se alcanza, se declara error
    //en toda la unidad de trabajo y se dispara el
    //asimilador. Previene el fallo de la aplicación
    wu.max_error_results = 5;

    //wu.max_total_results: Si el número de resultados de una
    //unidad de trabajo es mayor que este número, se declara
    //error de la unidad de trabajo para prevenir fallo del
    //núcleo del cliente.
    wu.max_total_results = 5;

    //wu.max_success_results: Si el número de resultados
    //exitosos excede este número pero no hay concordancia,
    //se declara error de la unidad de trabajo para evitar
    //unidades de trabajo que producen resultados no
    //determinísticos
    wu.max_success_results = 5;

    //wu.rsc_fpops_est: Es el número estimado de operaciones
    //de punto flotante que se requieren para completar un
    //trabajo y se utiliza para estimar cuánto tiempo tardará
    //un trabajo en realizarse en un host dado
    wu.rsc_fpops_est = 1e10;
```

```
    //wu.rsc_fpops_bound: Es el número maximo de operaciones
    //de punto flotante que se requieren para completar un
    //trabajo. Si se excede este número, se aborta el
    //trabajo.
    wu.rsc_fpops_bound = 1e11;

    //wu.rsc_memory_bound: Cantidad de RAM minima para
    //realizar el trabajo. El trabajo solo se envia a
    //clientes con al menos esta cantidad de RAM disponible
    wu.rsc_memory_bound = 1e8;

    //wu.rsc_disk_bound: Cantidad necesaria de espacio en
    //disco duro para realizar el trabajo, por lo tanto, se
    //enviará a clientes que tengan minimo esta cantidad de
    //disco duro disponible.
    wu.rsc_disk_bound = 1e8;

    //wu.delay_bound:
    wu.delay_bound = 7*86400;

    //En este ejemplo se crea solo una unidad de trabajo.
    create_work(
        wu,
        wu_template,
"templates/output_template.xml",
"templates/output_template.xml",
        infiles,
        1,
        config
    );
}
```

VALIDADOR DE EJEMPLO

```
/*****************************************************
* Ejemplo obtenido de                                *
* https://boinc.berkeley.edu/trac/wiki/ValidationSimple*
*****************************************************/

#include <string>
#include <vector>
#include <math.h>
#include "error_numbers.h"
#include "boinc_db.h"
#include "sched_util.h"
#include "validate_util.h"
using std::string;
using std::vector;

struct DATA {
    int i;
    double x;
};

int init_result(RESULT const & result, void*& data) {
    FILE* f;
    OUTPUT_FILE_INFO fi;
    int i, n, retval;
    double x;

    retval = get_output_file_path(result, fi.path);
    if (retval) return retval;
    retval = try_fopen(fi.path.c_str(), f, "r");
    if (retval) return retval;
    n = fscanf(f, "%d %f", &i, &x);
    fclose(f);
    if (n != 2) return ERR_XML_PARSE;
    DATA* dp = new DATA;
    dp->i = i;
    dp->x = x;
    data = (void*) dp;
    return 0;
}
```

```
int compare_results(
    RESULT& r1, void* _data1, RESULT const& r2, void* _data2,
bool& match
) {
    DATA* data1 = (DATA*)_data1;
    DATA* data2 = (DATA*)_data2;
    match = true;
    if (data1->i != data2->i) match = false;
    if (fabs(data1->x - data2->x) > 0.01) match = false;
    return 0;
}

int cleanup_result(RESULT const& r, void* data) {
    if (data) delete (DATA*) data;
    return 0;
}
```

ASIMILADOR DE EJEMPLO

```
/* Ejemplo obtenido de BOINC/sched/sample_assimilator.cpp */
#include <vector>
#include <string>
#include <cstdlib>

#include "boinc_db.h"
#include "error_numbers.h"
#include "filesys.h"
#include "sched_msgs.h"
#include "validate_util.h"
#include "sched_config.h"

using std::vector;
using std::string;

int write_error(char* p) {
     static FILE* f = 0;
     if (!f) {
     f = fopen(config.project_path("sample_results/errors"),
          "a");
     if (!f) return ERR_FOPEN;
     }
     fprintf(f, "%s", p);
     fflush(f);
     return 0;
}

int assimilate_handler(

     //variables de referencia a memoria para unidades de
     //trabajo, contenedor de resultados, resultado canonico
//(si existe)
     WORKUNIT& wu, vector<RESULT>& /*results*/,
     RESULT& canonical_result
) {
     int retval;
```

```
char buf[1024];
unsigned int i;

retval =
boinc_mkdir(config.project_path("sample_results"));
if (retval) return retval;

if (wu.canonical_resultid) {
vector<OUTPUT_FILE_INFO> output_files;
const char *copy_path;
get_output_file_infos(canonical_result, output_files);
unsigned int n = output_files.size();
bool file_copied = false;
for (i=0; i<n; i++) {
OUTPUT_FILE_INFO& fi = output_files[i];
if (n==1) {
copy_path =
          config.project_path("sample_results/%s",
          wu.name);
} else {
copy_path =
          config.project_path("sample_results/%s_%d",
          wu.name, i);
}
retval = boinc_copy(fi.path.c_str() , copy_path);
if (!retval) {
file_copied = true;
}
}
if (!file_copied) {
copy_path = config.project_path(
"sample_results/%s_%s", wu.name,
          "no_output_files"
);
FILE* f = fopen(copy_path, "w");
fclose(f);
}
} else {
sprintf(buf, "%s: 0x%x\n", wu.name, wu.error_mask);
return write_error(buf);
```

```
      }
      return 0;
}
```

ARCHIVO DE DESCRIPCIÓN DE TRABAJO PARA LA APLICACIÓN SERPENT-WRAPPER

```
<job_desc>
      <task>
            <application>sss2</application>
            <command_line>in</command_line>
      </task>
      <unzip_input>
            <zipfilename>in.zip</zipfilename>
      </unzip_input>
      <zip_output>
            <zipfilename>out.zip</zipfilename>
            <filename>[in][a-z\.\_]*</filename>
      </zip_output>
</job_desc>
```

ARCHIVO DE DESCRIPCIÓN DE LA VERSIÓN DE LA APLICACIÓN SERPENT-WRAPPER

```
<version>
    <file>
      <physical_name>wrapper</physical_name>
      <main_program/>
    </file>
    <file>
      <physical_name>sss2_1.0_x86_64-pc-linux-gnu</physical_name>
      <logical_name>sss2</logical_name>
    </file>
    <file>
      <physical_name>sss2_job_1.0.xml</physical_name>
      <logical_name>job.xml</logical_name>
    </file>
</version>
```

GENERADOR DE TRABAJO DE LA APLICACIÓN SERPENT-WRAPPER

```
#include <string>
#include <cstring>
#include <stdio.h>

#include "filesys.h"
#include "boinc_db.h"
#include "error_numbers.h"
#include "backend_lib.h"
#include "parse.h"
#include "util.h"

#include "sched_config.h"
#include "sched_util.h"
#include "sched_msgs.h"

#define CUSHION 5
  // maintain at least this many unsent tasks (results)
#define REPLICATION_FACTOR  1
  // generate as much tasks for every job (workunit) created
#define ARCHIVOS "../archivos"
  // The directory where the input files are stored
  // NO trailing slash!
// Search for TODO within this file to see more configuration
options

// globals
char* wu_template;
DB_APP app;
int start_time;
int seqno;

// create one new job
//
int make_job(const char* filename) {
    DB_WORKUNIT wu;
    char wu_name[256], dst_path[256], src_path[256];
    const char* infiles[1];
    int retval;

    // make a unique name and move input file into download dir
    sprintf(wu_name, "sss2_%d_%d", start_time, seqno++); // TODO:
change this if you want to generate the workunitname using the
input file name
    log_messages.printf(MSG_DEBUG,
```

```
            "Archivo %s encontrado, creando trabajo %s\n",
filename, wu_name
    );
    retval = config.download_path(filename, dst_path);
    if (retval) return retval;
    sprintf(src_path, "%s/%s", ARCHIVOS, filename);
    log_messages.printf(MSG_DEBUG,
            "Moviendo archivo %s a %s\n", src_path, dst_path
    );
    retval = rename(src_path, dst_path);
    if (retval) {
        log_messages.printf(MSG_CRITICAL,
            "rename: %d, errno is %d\n", retval, errno
        );
        return retval;
    }

    // Fill in the job parameters
    //
    wu.clear();
    wu.appid = app.id;
    strcpy(wu.name, wu_name);
    wu.rsc_fpops_est = 1e12;
    wu.rsc_fpops_bound = 1e12;
    wu.rsc_memory_bound = 1e8;
    wu.rsc_disk_bound = 1e8;
    wu.delay_bound = 86400;
    wu.min_quorum = REPLICATION_FACTOR;
    wu.target_nresults = REPLICATION_FACTOR;
    wu.max_error_results = REPLICATION_FACTOR*2;
    wu.max_total_results = REPLICATION_FACTOR*4;
    wu.max_success_results = REPLICATION_FACTOR*2;
    infiles[0] = filename;

    // Register the job with BOINC
    //
    // TODO: change result template here if needed
    return create_work(
        wu,
        wu_template,
        "templates/sss2_out",
        "../templates/sss2_out",
        infiles,
        1,
        config
    );
}

void main_loop() {
    std::string file;
```

```
    int retval;

    while (1) {
        check_stop_daemons();
        long n;
        retval = count_unsent_results(n, 0);
        if (n > CUSHION) {
            sleep(60);
        } else {
            DirScanner dirscan(ARCHIVOS);
            // check if there are files in INPUT_FOLDER
            if (dirscan.scan(file)) {
                // files found, now create work for them
                int njobs = (CUSHION-n)/REPLICATION_FACTOR;
                log_messages.printf(MSG_DEBUG,
                    "Haciendo %d trabajos\n", njobs
                );
                for (int i=0; i<njobs; i++) {
                    retval = make_job(file.c_str());
                    if (retval) {
                        log_messages.printf(MSG_CRITICAL,
                            "No se pudo hacer el trabajo: %d\n",
retval
                        );
                        exit(retval);
                    }
                    if (!dirscan.scan(file)) break; // no more
files but CUSHION not yet reached
                }
            } else {
                // no more files found, wait some time
                log_messages.printf(MSG_DEBUG,
                        "No hay mas archivos en: %s\n", ARCHIVOS
                );
                sleep(60); //increase this time if you can't
supply a steady stream of input files
            }
            // Now sleep for a few seconds to let the transitioner
            // create instances for the jobs we just created.
            // Otherwise we could end up creating an excess of
jobs.
            sleep(5);
        }
    }
}

int main(int argc, char** argv) {
    int i, retval;
```

```
    for (i=1; i<argc; i++) {
        if (!strcmp(argv[i], "-d")) {
            log_messages.set_debug_level(atoi(argv[++i]));
        } else {
            log_messages.printf(MSG_CRITICAL,
                "bad cmdline arg: %s", argv[i]
            );
        }
    }

    if (config.parse_file("..")) {
        log_messages.printf(MSG_CRITICAL,
            "can't read config file\n"
        );
        exit(1);
    }

    retval = boinc_db.open(
        config.db_name, config.db_host, config.db_user,
config.db_passwd
    );
    if (retval) {
        log_messages.printf(MSG_CRITICAL, "can't open db\n");
        exit(1);
    }
    // TODO: change appname for which the work is generated
    if (app.lookup("where name='sss2'")) {
        log_messages.printf(MSG_CRITICAL, "can't find app\n");
        exit(1);
    }
    // TODO: change workunit template if needed
    if (read_file_malloc("../templates/sss2_in", wu_template)) {
        log_messages.printf(MSG_CRITICAL, "can't read WU
template\n");
        exit(1);
    }

    start_time = time(0);
    seqno = 0;

    log_messages.printf(MSG_NORMAL, "Starting\n");

    main_loop();
}
```

APLICACIÓN DE TRABAJO DEL PROYECTO EULER

```
#include <stdio.h>
#include "boinc_api.h"
#include "filesys.h"
#include "mfile.h"
#include <mpfr.h>
#include <sys/time.h>

void potenciab(mpfr_t pot, mpfr_t base, unsigned long int exp){
      unsigned long int i;
      fprintf(stderr, "Calculando la potencia %lu de 1/", exp);
      mpfr_fprintf(stderr, "%.1Rf", base);
      fprintf(stderr,"\n");
      mpfr_set_d(pot, 1.0, MPFR_RNDD);

      if(exp != 0)
            for(i=1; i<=exp; i++){
                  mpfr_div(pot, pot, base, MPFR_RNDD);
            }
}

void coefBin(mpfr_t coef, unsigned long int n, unsigned long int
m){
      unsigned long int i;
      mpfr_t f;

      mpfr_init2(f,500);
      mpfr_set_d(f, 1.0, MPFR_RNDD);

      //Sabemos que n siempre serÃ¡ mayor que m y que (n-m)
      fprintf(stderr, "Calculando el coeficiente binario...\n");
      mpfr_set_d(coef, 1.0, MPFR_RNDD);

      if(n!=0)
                  for(i=1; i<=n; i++){
                          //printf("\n%lu fact = ", i);
                          mpfr_mul_ui(f, f, i, MPFR_RNDU);
                          //mpfr_printf("%.128Rf", f);

                  if(i == (n-m)|| i == m){
                        //printf("%lu encontrado\n", i);
                        mpfr_div(coef, coef, f, MPFR_RNDD);
                        //mpfr_printf("%.128Rf", coef);
                        //printf("\n");
                  }
                  }
```

```
	mpfr_mul(coef, coef, f, MPFR_RNDD);
	//mpfr_printf("%.128Rf", coef);
	//printf("\n");

	mpfr_clear(f);
}

void ktermino(FILE *entrada, FILE *salida){
	unsigned long int ulint_k, ulint_n;
	   struct timeval inicio, fin;
	double useg;

	   gettimeofday(&inicio, NULL);

	if(!feof(entrada)){
		fscanf(entrada, "%lu,%lu", &ulint_n, &ulint_k);
		fprintf(stderr,"Obtenidos: n = %lu, k = %lu\n", ulint_n,
ulint_k);
	}
	else{
		fprintf(stderr, "ERROR!! el archivo esta vacÃ-o\n");
	}

	mpfr_t termino, bk, pbk, back;

	mpfr_init2(termino, 500);
	mpfr_init2(bk, 500);
	mpfr_init2(pbk, 500);
	mpfr_init2(back, 500);

	ulint_k -= 1;

	coefBin(termino, ulint_n, ulint_k);
	mpfr_set_ui(bk, ulint_n, MPFR_RNDD);
	potenciab(pbk, bk, ulint_k);
	//mpfr_printf("%.4096Rf", pbk);
	//printf("\n");

	mpfr_mul(termino, termino, pbk, MPFR_RNDD);
	fprintf(stderr, "\n\nEl k termino es: \n");
	mpfr_fprintf(stderr, "%.4096Rf", termino);
	mpfr_out_str(salida, 10, 8192, termino, MPFR_RNDD);
	fprintf(stderr, "\n");

	mpfr_clear(bk);
	mpfr_clear(pbk);
	mpfr_clear(termino);

	gettimeofday(&fin, NULL);
```

```
	useg = (double)((fin.tv_usec - inicio.tv_usec) + (fin.tv_sec
- inicio.tv_sec)*1000000);
	fprintf(stderr, "Tiempo de ejecuciÃ³n: %f [us]\n", useg);
}

int main(){
	FILE *in, *out;
	int retval;
	char input_path[512], output_path[512], chkpt_path[512],
buf[256];
	double fsize, fd;

	retval = boinc_init();
	if(retval){
		fprintf(stderr, "ERROR!!! %s boinc_init regreso %d\n",
boinc_msg_prefix(buf, sizeof(buf)), retval);
		exit(retval);
	}
	fprintf(stderr, "App iniciada\n");

	boinc_resolve_filename("in", input_path, sizeof(input_path));
	fprintf(stderr,"El archivo de entrada esta en:
%s\n",input_path);
	in = boinc_fopen(input_path, "r");

	if(!in){
		fprintf(stderr,"ERROR!! %s BOINC no encontro el archivo
en %s",boinc_msg_prefix(buf, sizeof(buf)), input_path);
		exit(-1);
	}

	file_size(input_path, fsize);

	boinc_resolve_filename("out", output_path,
sizeof(output_path));
	out = boinc_fopen(output_path,"wb");
	if(!out){
			fprintf(stderr,"ERROR!! %s BOINC no pudo crear el
archivo en %s\n", boinc_msg_prefix(buf, sizeof(buf)),
output_path);
			exit(-1);
	   }

	ktermino(in, out);

	boinc_finish(0);
}
```

SCRIPT PARA MOVER LAS LIBRERÍAS SERPENT-WRAPPER

```
#!/bin/bash

S1="jeff31"
for i in $(ls ../projects/zeus.fi-b.unam.mx_sss2); do
    echo $i
    if [ "$S1"!="$i" ];
     then
     x=1
     echo $x
    fi
    done

    if [ $x=1 ];
     then
     mkdir ../projects/zeus.fi-b.unam.mx_sss2/jeff31
     mkdir ../projects/zeus.fi-b.unam.mx_sss2/jeff31/acedata

     for k in `awk '1' librerias.txt`
         do
         echo $k
         mv $k ../projects/zeus.fi-b.unam.mx_sss2/jeff31/acedata/
         done
    fi
```

GENERADOR DE TRABAJO DEL PROYECTO EULER

```
// This file is part of BOINC.
// http://boinc.berkeley.edu
// Copyright (C) 2008 University of California
//
// BOINC is free software; you can redistribute it and/or modify
it
// under the terms of the GNU Lesser General Public License
// as published by the Free Software Foundation,
// either version 3 of the License, or (at your option) any later
version.
//
// BOINC is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
// See the GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General
Public License
// along with BOINC.  If not, see <http://www.gnu.org/licenses/>.

// sample_work_generator: example BOINC work generator.
//
// --app name               app name (default example_app)
// --in_template_file       input template file (default
example_app_in)
// --out_template_file      output template file (default
example_app_out)
// -d N                     log verbosity level (0..4)
// --help                   show usage
// --version                show version
//
// - Runs as a daemon, and creates an unbounded supply of work.
//   It attempts to maintain a "cushion" of 100 unsent job
instances
//   for the given app.
//   (your app may not work this way; e.g. you might create work
in batches)
// - Creates a new input file for each job;
//   the file (and the workunit names) contain a timestamp
//   and sequence number, so they're unique.
//
// This is an example - customize for your needs

#include <sys/param.h>
#include <unistd.h>
#include <cstdlib>
```

```
#include <string>
#include <cstring>

#include "backend_lib.h"
#include "boinc_db.h"
#include "error_numbers.h"
#include "filesys.h"
#include "parse.h"
#include "str_replace.h"
#include "str_util.h"
#include "svn_version.h"
#include "util.h"

#include "sched_config.h"
#include "sched_util.h"
#include "sched_msgs.h"

#define CUSHION 10
    // maintain at least this many unsent results
#define REPLICATION_FACTOR  1
    // number of instances of each job

//#define N 100

const char* app_name = "euler";
const char* in_template_file = "euler_in";
const char* out_template_file = "euler_out";

char* in_template;
DB_APP app;
int start_time;
unsigned int N;
unsigned long int ne;
int seqno;

// create one new job
//
int make_job(unsigned long int kterm) {
    DB_WORKUNIT wu;
    char name[256], path[MAXPATHLEN];
    const char* infiles[1];
    int retval;

    ne = N;

    // make a unique name (for the job and its input file)
    //
    sprintf(name, "%s_%lu_%d_%d", app_name, kterm, start_time,
seqno++);
```

```
    // Create the input file.
    // Put it at the right place in the download dir hierarchy
    //
    retval = config.download_path(name, path);
    if (retval) return retval;
    FILE* f = fopen(path, "w");
    if (!f) return ERR_FOPEN;
    fprintf(f, "%lu,%lu", ne, kterm);
    fclose(f);

    // Fill in the job parameters
    //
    wu.clear();
    wu.appid = app.id;
    safe_strcpy(wu.name, name);
    wu.rsc_fpops_est = 1e12;
    wu.rsc_fpops_bound = 1e14;
    wu.rsc_memory_bound = 1e8;
    wu.rsc_disk_bound = 1e8;
    wu.delay_bound = 86400;
    wu.min_quorum = REPLICATION_FACTOR;
    wu.target_nresults = REPLICATION_FACTOR;
    wu.max_error_results = REPLICATION_FACTOR*4;
    wu.max_total_results = REPLICATION_FACTOR*8;
    wu.max_success_results = REPLICATION_FACTOR*4;
    infiles[0] = name;

    // Register the job with BOINC
    //
    sprintf(path, "templates/%s", out_template_file);
    return create_work(
        wu,
        in_template,
        path,
        config.project_path(path),
        infiles,
        1,
        config
    );
}

void main_loop() {
    int retval;
    unsigned long int k = 0;

    while (1) {
        check_stop_daemons();
        long n;
        retval = count_unsent_results(n, app.id);
```

```
        if (retval) {
            log_messages.printf(MSG_CRITICAL,
                "count_unsent_jobs() failed: %s\n",
boincerror(retval)
            );
            exit(retval);
        }
        if (n > CUSHION) {
            daemon_sleep(10);
        } else {
            int njobs = (CUSHION-n)/REPLICATION_FACTOR;
            log_messages.printf(MSG_DEBUG,
                "Making %d jobs\n", njobs
            );
            for (int i=0; i<njobs; i++) {
                k++;
                if(k<=N){
                retval = make_job(k);
                if (retval) {
                          log_messages.printf(MSG_CRITICAL,"can't
make job: %s\n", boincerror(retval));
                          exit(retval);
                }
                }else{
                     log_messages.printf(MSG_CRITICAL,"Ya se
crearon %lu tareas, durmiendo generador de trabajo\n", ne);
                     daemon_sleep(50);
                }
        }
            // Wait for the transitioner to create instances
            // of the jobs we just created.
            // Otherwise we'll create too many jobs.
            //
            double now = dtime();
            while (1) {
                daemon_sleep(5);
                double x;
                retval = min_transition_time(x);
                if (retval) {
                    log_messages.printf(MSG_CRITICAL,
                        "min_transition_time failed: %s\n",
boincerror(retval)
                    );
                    exit(retval);
                }
                if (x > now) break;
            }
        }
    }
}
```

```
void usage(char *name) {
    fprintf(stderr, "This is an example BOINC work generator.\n"
        "This work generator has the following properties\n"
        "(you may need to change some or all of these):\n"
        "  It attempts to maintain a \"cushion\" of 100 unsent job
instances.\n"
        "  (your app may not work this way; e.g. you might create
work in batches)\n"
        "- Creates work for the application \"example_app\".\n"
        "- Creates a new input file for each job;\n"
        "  the file (and the workunit names) contain a
timestamp\n"
        "  and sequence number, so that they're unique.\n\n"
        "Usage: %s [OPTION]...\n\n"
        "Options:\n"
        "  [ --app X                Application name (default:
example_app)\n"
        "  [ --in_template_file     Input template (default:
example_app_in)\n"
        "  [ --out_template_file    Output template (default:
example_app_out)\n"
        "  [ -d X ]                 Sets debug level to X.\n"
        "  [ -h | --help ]          Shows this help text.\n"
        "  [ -v | --version ]       Shows version information.\n"
     "  [--neuler ]           NÃºmero de terminos.\n",
        name
    );
}

int main(int argc, char** argv) {
    int i, retval;
    char buf[256];

    for (i=1; i<argc; i++) {
        if (is_arg(argv[i], "d")) {
            if (!argv[++i]) {
                log_messages.printf(MSG_CRITICAL, "%s requires an
argument\n\n", argv[--i]);
                usage(argv[0]);
                exit(1);
            }
            int dl = atoi(argv[i]);
            log_messages.set_debug_level(dl);
            if (dl == 4) g_print_queries = true;
        } else if (!strcmp(argv[i], "--app")) {
            app_name = argv[++i];
        } else if (!strcmp(argv[i], "--in_template_file")) {
            in_template_file = argv[++i];
```

```
        } else if (!strcmp(argv[i], "--out_template_file")) {
            out_template_file = argv[++i];
        } else if (is_arg(argv[i], "h") || is_arg(argv[i],
"help")) {
            usage(argv[0]);
            exit(0);
        } else if (is_arg(argv[i], "v") || is_arg(argv[i],
"version")) {
            printf("%s\n", SVN_VERSION);
            exit(0);
     } else if (!strcmp(argv[i], "--neuler")) {
            N = std::stoi(argv[++i],nullptr);
            exit(0);
        } else {
            log_messages.printf(MSG_CRITICAL, "unknown command
line argument: %s\n\n", argv[i]);
            usage(argv[0]);
            exit(1);
        }
    }

    retval = config.parse_file();
    if (retval) {
        log_messages.printf(MSG_CRITICAL,
            "Can't parse config.xml: %s\n", boincerror(retval)
        );
        exit(1);
    }

    retval = boinc_db.open(
        config.db_name, config.db_host, config.db_user,
config.db_passwd
    );
    if (retval) {
        log_messages.printf(MSG_CRITICAL, "can't open db\n");
        exit(1);
    }

    snprintf(buf, sizeof(buf), "where name='%s'", app_name);
    if (app.lookup(buf)) {
        log_messages.printf(MSG_CRITICAL, "can't find app %s\n",
app_name);
        exit(1);
    }

    snprintf(buf, sizeof(buf), "templates/%s", in_template_file);
    if (read_file_malloc(config.project_path(buf), in_template)) {
        log_messages.printf(MSG_CRITICAL, "can't read input
template %s\n", buf);
        exit(1);
```

```
    }

    start_time = time(0);
    seqno = 0;

    log_messages.printf(MSG_NORMAL, "Starting\n");

    main_loop();
}
```

VALIDADOR DEL PROYECTO EULER

```
// This file is part of BOINC.
// http://boinc.berkeley.edu
// Copyright (C) 2008 University of California
//
// BOINC is free software; you can redistribute it and/or modify
it
// under the terms of the GNU Lesser General Public License
// as published by the Free Software Foundation,
// either version 3 of the License, or (at your option) any later
version.
//
// BOINC is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
// See the GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General
Public License
// along with BOINC.  If not, see <http://www.gnu.org/licenses/>.

// A sample validator that accepts all results


#include <string>
#include <stdlib.h>
#include "sched_util.h"
#include "sched_msgs.h"
#include "error_numbers.h"
#include "boinc_db.h"
#include "sched_util.h"
#include "validate_util.h"
#include "validate_util2.h"
#include <mpfr.h>

#define ERR_EULER_NZERO -300

struct DATA{
    unsigned long int k;
    mpfr_t kterm;
};

int init_result(RESULT& result, void*& data) {
    int retval;
    FILE* f;
    OUTPUT_FILE_INFO fi;
    unsigned long int k;
    char* out_f;
```

```
    char* token;
    mpfr_t kterm;

    mpfr_init2(kterm, 500);

    retval = get_output_file_path(result, fi.path);
    if (retval) return retval;

    retval = try_fopen(fi.path.c_str(), f, "r");
    if (retval) return retval;
    if(!mpfr_inp_str(kterm, f, 10, MPFR_RNDD)){
     mpfr_clear(kterm);
     fclose(f);
     log_messages.printf(MSG_CRITICAL,
            "[RESULT#%lu %s] check_set: Imposible leer el tÃ©rmino
del archivo.\n",
            result.id, result.name
        );
     return ERR_FOPEN;
    }
    fclose(f);

    if(mpfr_sgn(kterm) <= 0){
     mpfr_clear(kterm);
     log_messages.printf(MSG_CRITICAL,
            "[RESULT#%lu %s] check_set: TÃ©rmino invÃ¡lido (menor
o igual a cero)\n",
            result.id, result.name
        );
     return ERR_EULER_NZERO;
    }
    mpfr_init_set(dp->kterm, kterm, MPFR_RNDD);
    mpfr_clear(kterm);
    data = (void*) dp;
    return 0;
}

int compare_results(RESULT&, void*, RESULT const&, void*, bool&
match) {
    match = true;
    return 0;
}

int cleanup_result(RESULT const& r, void* data) {
    if (data) delete (DATA*) data;
    return 0;
}

const char *BOINC_RCSID_f3a7a34795 = "$Id$";
```

ASIMILADOR DEL PROYECTO EULER

```
// This file is part of BOINC.
// http://boinc.berkeley.edu
// Copyright (C) 2015 University of California
//
// BOINC is free software; you can redistribute it and/or modify
it
// under the terms of the GNU Lesser General Public License
// as published by the Free Software Foundation,
// either version 3 of the License, or (at your option) any later
version.
//
// BOINC is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
// See the GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General
Public License
// along with BOINC.  If not, see <http://www.gnu.org/licenses/>.

// A sample assimilator that:
// 1) if success, copy the output file(s) to a directory
// 2) if failure, append a message to an error log

#include <vector>
#include <string>
#include <cstdlib>

#include "boinc_db.h"
#include "error_numbers.h"
#include "filesys.h"
#include "sched_msgs.h"
#include "validate_util.h"
#include "sched_config.h"
#include "assimilate_handler.h"
#include <mpfr.h>

using std::vector;
using std::string;

const char* outdir = "sample_results";

int write_error(char* p) {
    static FILE* f = 0;
    if (!f) {
        char path[1024];
```

```
        sprintf(path, "%s/errors", outdir);
        f = fopen(config.project_path(path), "a");
        if (!f) return ERR_FOPEN;
    }
    fprintf(f, "%s", p);
    fflush(f);
    return 0;
}

int assimilate_handler_init(int argc, char** argv) {
    for (int i=1; i<argc; i++) {
        if (!strcmp(argv[i], "--outdir")) {
            outdir = argv[++i];
        } else {
            fprintf(stderr, "bad arg %s\n", argv[i]);
        }
    }
    return 0;
}

void assimilate_handler_usage() {
    // describe the project specific arguments here
    fprintf(stderr,
        "    Custom options:\n"
        "    [--outdir X]  output dir for result files\n"
    );
}

int assimilate_handler(
    WORKUNIT& wu, vector<RESULT>& /*results*/, RESULT&
canonical_result
) {
    int retval;
    char buf[1024];
    char archivo[1024];
    unsigned int i;

    FILE *f;

    mpfr_t e;
    mpfr_t coef;

    mpfr_init2(e, 500);
    mpfr_init2(coef, 500);

    retval = boinc_mkdir(config.project_path(outdir));
    if (retval) return retval;

    if (wu.canonical_resultid) {
        vector<OUTPUT_FILE_INFO> output_files;
```

```
        const char *copy_path;
        get_output_file_infos(canonical_result, output_files);
        unsigned int n = output_files.size();
        bool file_copied = false;
        for (i=0; i<n; i++) {
            OUTPUT_FILE_INFO& fi = output_files[i];
            if (n==1) {
                sprintf(buf, "%s/%s", outdir, wu.name);
                sprintf(archivo, "%s", wu.name);
            } else {
                sprintf(buf, "%s/%s_%d", outdir, wu.name, i);
            }
            copy_path = config.project_path(buf);
            retval = boinc_copy(fi.path.c_str() , copy_path);
            if (!retval) {
                file_copied = true;
            }
        }
        if (!file_copied) {
            sprintf(buf, "%s/%s_no_output_files", outdir,
wu.name);
            copy_path = config.project_path(buf);
            FILE* f = fopen(copy_path, "w");
            if (!f) return ERR_FOPEN;
            fclose(f);
        }
    } else {
        sprintf(buf, "%s: 0x%x\n", wu.name, wu.error_mask);
        return write_error(buf);
    }

/***********************************************/
/* Manejo del contenido de los archivos        */
/***********************************************/

    f = boinc_fopen("e", "rb");
    if(f){
        mpfr_inp_str(e, f, 10, 8192, MPFR_RNDD);
        fclose(f);
    }
    else{
        mpfr_set_d(e, 0.0, MPFR_RNDD);
    }

    f = boinc_fopen(archivo, "rb");

    if(f){
        mpfr_inp_str(coef, f, 10, 8192, MPFR_RNDD);
        fclose(f);
```

```
    mpfr_add(e, e, coef, MPFR_RNDD);
f = boinc_fopen("e", "wb");
if(f) mpfr_out_str(f, 10, 8192, e, MPFR_RNDD);
else write_error("Error al abrir el archivo");
fclose(f);
mpfr_clears(e, archivo);
return 0;
```

```
}
```

REFERENCIAS

[1]. Lawrence Livermore National Laboratory. (2014). Introduction to Parallel Computing. California, E.U. Recuperado el 11 de diciembre de 2014 de: https://computing.llnl.gov/tutorials/parallel_comp/

[2]. Rauber, Th, Rünger, G. (2013). Cap 2. Parallel Programming for Multicore and Cluster Systems. (pp 9 - 103) Recuperado el 11 de diciembre de 2014 de: http://www.springer.com/cda/content/document/cda_downloaddocument/9783642378003-c2.pdf?SGWID=0-0-45-1404786-p175291633

[3]. Tanenbaum, A. (1995) Distributed Operating Systems, Prentice Hall, New Jersey, USA.

[4]. Intel Corporation (2016). California, E.U. Recuperado el 24 de julio de 2016 de: https://www.intel.com/pressroom/kits/upcrc/ParallelComputing_backgrounder.pdf

[5]. International Business Machines Corp. (2016). 7030 Data Processing System. New York, E.U. Recuperado el 26 de julio de 2016 de: https://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP7030.html

[6]. Lawrence Livermore National Laboratory. (2016). The Remington Rand UNIVAC LARC. California, E.U. Recuperado el 19 de agosto de 2016 de: http://www.computer-history.info/Page4.dir/pages/LARC.dir/LARC.Cole.html

[7]. Lawrence Livermore National Laboratory. (2016). A history of LLNL Computing. California, E.U. Recuperado el 19 de agosto de 2016 de: http://computation.llnl.gov/history

[8]. Computer History Museum Revolution. (2016) Establishing a Pattern: Von Neumann at the IAS. Recuperado el 21 de agosto de 2016 de: http://www.computerhistory.org/revolution/supercomputers/10/28

[9]. Eckert, J.P. (1899). UNIVAC LARC, The next Step Computer Design. (pp 16 - 17) Recuperado el 21 de agosto de 2016 de: https://www.computer.org/csdl/proceedings/afips/1956/5049/00/50490016.pdf

[10]. Kempt, J. (2015). Atlas, the UK's supercomputer. Recuperado el 23 de agosto de 2016 de: https://www.linuxvoice.com/atlas-the-uks-supercomputer/

[11]. Computer History Museum Revolution. (2016). The "Manchester TC" transistor computer. Recuperado el 5 de septiembre de 2016 de: http://www.computerhistory.org/revolution/digital-logic/12/273/1371

[12]. McAllister, N. (08/12/2012). Behold ATLAS, the fastest computer of 50 years ago. Recuperado el 5 de septiembre de 2016 de: http://www.theregister.co.uk/2012/12/08/ferranti_atlas_50th_birthday/

[13]. Dally, B. (2002). EE4482C: Advanced Computer Organization. Lecture 11: Vector Arquitectures (pp 1-3). Recuperado el 5 de septiembre de 2016 de: http://cva.stanford.edu/classes/ee482s/scribed/lect11.pdf

[14]. Welch, E. Evans, J. (13/05/2013). Vector and SIMD prossesors. Recuperado el 5 de septiembre de 2016 de: http://meseec.ce.rit.edu/756-projects/spring2013/2-2.pdf

[15]. Westinghouse Defence and Space Center. (04/11/1963). Solomon Project Technical Memorandum No. 25: Solomon II Physical Characteristics (pp 1-7). Recuperado el 9 de septiembre de 2016 de: https://archive.org/stream/bitsavers_westinghoumonProjectTechnicalMemorandum25SOLOMONII_1563319/15090B_Solomon_Project_Technical_Memorandum_25_SOLOMON_II_Physical_Characteristics_Nov63#page/n0/mode/2up

[16]. Barnes G.H. Brown, R.M. (08/08/1968) The ILLIAC IV Computer. (pp 1- 4) Recuperado el 9 de septiembre de 2016 de: http://gordonbell.azurewebsites.net/tcmwebpage/timeline/chap27_and20cs2_illiaciv_cs1.pdf

[17]. Burroughs Corporation. (1974) ILLIAC IV (pp 1-5). Recuperado el 9 de septiembre de 2016 de: http://archive.computerhistory.org/resources/text/Burroughs/Burroughs.ILLIACIV.1974.102624911.pdf

[18]. University of Edimburgh. School of Informatics. (1996). High Performance Computer Architecture. The CDC Star-100. Recuperado el 18 de septiembre

de 2016 de: http://homepages.inf.ed.ac.uk/cgi/rni/comp-arch.pl?Vect/star100.html,Vect/star100cpu-f.html,Vect/menu-cyb.html

[19]. University of Edimburgh. School of Informatics. (1996). High Performance Computer Architecture. The Cray-1. Recuperado el 18 de septiembre de 2016 de: http://homepages.inf.ed.ac.uk/cgi/rni/comp-arch.pl?Vect/cray1.html,Vect/cray1-cpu.gif,Vect/menu-cr1.html

[20]. Yeo, C.S. Buyya, R. et al. (2006). Cluster Computing: High-Performance, High-Availability, and High-Throughput Processing on a Network of Computers. Recuperado el 19 de septiembre de 2016 de: http://www.cloudbus.org/papers/ic_cluster.pdf

[21]. Chandra, R., Dagum, L. (2001). Parallel Programming in OpenMP. California, USA. Academic Press.

[22]. Lawrence Livermore National Laboratory. (2014). POSIX Thread Programming. California, E.U. Recuperado el 19 de septiembre de 2016 de: https://computing.llnl.gov/tutorials/pthreads/

[23]. Lawrence Livermore National Laboratory. (2014). Message Passing Interface (MPI). California, E.U. Recuperado el 19 de septiembre de 2016 de: https://computing.llnl.gov/tutorials/mpi/

[24]. Menchaca, R. García, F. (2001). Java RMI. Revista Digital Universitaria, Vol. 2, Recuperado el 19 de septiembre de 2016 de: http://www.revista.unam.mx/vol.2/num1/art3/

[25]. Erciyes, K. (2013). Distributed Graph Algorithms for Computer Networks. DOI: 10.1007/978-1-4471-5173-9

[26]. Wattenhofer, R. (2014). Principles of Distributed Computing. Recuperado el 19 de septiembre de 2016 de: http://dcg.ethz.ch/lectures/podc_allstars/lecture/podc.pdf

[27]. Ahn, N. Black, J. Effrat, J. (2001). Distributed Computing. History. Recuperado el 19 de septiembre de 2016 de: http://cs.stanford.edu/people/eroberts/courses/soco/projects/2001-02/distributed-computing/html/history.html

[28]. Lazalde, A. (2001). Historia de la tecnología: Creeper, el primer gusano informático. Recuperado el 19 de septiembre de 2016 de: https://hipertextual.com/2011/09/historia-tecnologica-creeper-primer-gusano-informatico

[29]. Hernández, N. (23/08/2015). El primer virus informático fue un experimento de laboratorio. Recuperado el 21 de septiembre de 2016 de: http://computerhoy.com/video/primer-virus-informatico-fue-experimento-laboratorio-32093

[30]. History of Computers. (2016). First computer virus of Bob Thomas. Recuperado el 21 de septiembre de 2016 de: http://history-computer.com/Internet/Maturing/Thomas.html

[31]. De León, C. Navarro, J. (2006). Cap 1. El Cómputo Paralelo y Distribuido. Desarrollo de un sistema de base de datos distribuida en un clúster de alto desempeño. Tesis de licenciatura. Universidad Nacional Autónoma de México. Recuperado el 21 de septiembre de 2016 de: http://132.248.9.195/pd2006/0606805/

[32]. Fedak, G. Germain, C. (13/12/2000). XtremeWeb: A generic global computing system. Recuperado el 23 de septiembre de 2016 de: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.24.5844&rep=rep1&type=pdf

[33]. Great Internet Mersenne Prime Search. (1996). History. Recuperado el 23 de septiembre de 2016 de: https://www.mersenne.org/various/history.php

[34]. distributed.net (18/04/2015). History & Timeline. Recuperado el 23 de septiembre de 2016 de: http://www.distributed.net/History

[35]. Dell EMC. RSA Laboratories. (2015). 3.6.4 What are RC5 and RC6? Recuperado el 23 de septiembre de 2016 de: https://www.emc.com/emc-plus/rsa-labs/standards-initiatives/rc5-and-rc6.htm

[36]. Dell EMC. RSA Laboratories. (2015). The RSA Laboratories Secret-Key Challenge. Recuperado el 25 de septiembre de 2016 de:

https://www.emc.com/emc-plus/rsa-labs/historical/the-rsa-laboratories-secret-key-challenge.htm

[37]. Dell EMC. RSA Laboratories. (2015). DES Challenge III. Recuperado el 25 de septiembre de 2016 de: https://www.emc.com/emc-plus/rsa-labs/historical/des-challenge-iii.htm

[38]. Standfor University. (2009). Folding@home: Lessons From Eight Years of Volunteer Distributed Computing. Recuperado el 25 de septiembre de 2016 de: mail.hicomb.org/papers/HICOMB2009-13.pdf

[39]. SETI Institute. (2017) Our work. Recuperado el 3 de enero de 2017 de: http://www.seti.org/node/647

[40]. SETI Institute. (2017) History of SETI. Recuperado el 3 de enero de 2017 de: http://www.seti.org/seti-institute/about-seti/press-materials/backgrounders/history-of-seti

[41]. Comunications of the ACM, Vol. 45, (2002) SETI@HOME: An experiment in public-resource computing. (pp 56-61). Recuperado el 3 de enero de 2017 de: http://setiathome.berkeley.edu/sah_papers/cacm.php

[42]. Kopela, E. Werthimer, D. (2001). Scientific Computing. SETI@HOME—MASSIVELY DISTRIBUTED COMPUTINGFOR SETI (pp 78-80). Recuperado el 3 de enero de 2017 de: http://setiathome.berkeley.edu/sah_papers/CISE.pdf

[43]. Oracle Help Center. (2017). Fusion Middleware Concepts Guide. Recuperado el 7 de enero de 2017 de: https://docs.oracle.com/cd/E21764_01/core.1111/e10103/intro.htm#ASCON113

[44]. University of California. (2014). BOINC. California, E.U. Recuperado el 4 de mayo de 2015 de: https://boinc.berkeley.edu/

[45]. University of California. (2014). Overview of BOINC. California, E.U. Recuperado el 4 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/BoincIntro

[46]. University of California. (2014). Why use BOINC? Recuperado el 4 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/WhyUseBoinc

[47]. Princeton University. (2007). PlanetLab. New Jersey, E.U. Recuperado el 26 de mayo de 2015 de: https://www.planet-lab.org/

[48]. Ferreira, L. et. al. (2002). Introduction to Grid Computing with Globus. IBM Redbook Series. Recuperado el 26 de mayo de 2015 de: http://ibm.com/redbooks.

[49]. University of Wisconsin-Madison (2015). Computing with HTCondor. Wisconsin, E.U. Recuperado el 26 de mayo de 2015 de: http://research.cs.wisc.edu/htcondor/

[50]. University of California. (2014). Setting up a BOINC server. Recuperado el 27 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/ServerIntro

[51]. University of California. (2014). Software prerequisites (Unix/Linux). Recuperado el 28 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/SoftwarePrereqsUnix

[52]. Ellinwood, J. DigitalOcean (2014) ¿Como instalar Linux, Apache, MySQL y PHP (LAMP) en Ubuntu14.04? Recuperado el 28 de mayo de 2015 de: https://www.digitalocean.com/community/tutorials/como-instalar-linux-apache-mysql-php-lamp-en-ubuntu-14-04-es

[53]. University of California. (2014). Building BOINC on Unix. Recuperado el 30 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/BuildSystem

[54]. University of California. (2014). Project creation cookbook. Recuperado el 30 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/CreateProjectCookbook

[55]. The Apache software foundation. (2016). Access control. Recuperado el 7 de enero de 2015 de: https://httpd.apache.org/docs/2.4/mod/access.html

[56]. University of California. (2014). The BOINC Database. Recuperado el 30 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/DataBase

[57]. University of California. (2014). Server directory structure. Recuperado el 30 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/ServerDirs

[58]. University of California. (2014). Staging input files. Recuperado el 30 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/JobStage

[59]. University of California. (2014). Archivo de configuración del proyecto. Recuperado el 30 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/ProjectConfigFile

[60]. University of California. (2014). Applications and app versions. Recuperado el 30 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/AppVersion

[61]. University of California. (2014). BOINC applications. Recuperado el 30 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/AppIntro.

[62]. University of California. (2014). The BOINC wrapper. Recuperado el 30 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/WrapperApp

[63]. University of California. (2014). Daemons. Recuperado el 30 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/ProjectDaemons

[64]. University of California. (2014). Work generators. Recuperado el 30 de mayo de 2015 de: https://boinc.berkeley.edu/trac/wiki/WorkGeneration

[65]. University of California. (2014). Validation. Recuperado el 1 de junio de 2015 de: https://boinc.berkeley.edu/trac/wiki/ValidationIntro

[66]. University of California. (2014). Handling completed jobs. Recuperado el 1 de junio de 2015 de: https://boinc.berkeley.edu/trac/wiki/AssimilateIntro

[67]. University of California. (2014). Creating application versions. Recuperado el 1 de junio de 2015 de: https://boinc.berkeley.edu/trac/wiki/AppVersionNew

[68]. University of California. (2014). xadd - tool for adding database items. Recuperado el 1 de junio de 2015 de: https://boinc.berkeley.edu/trac/wiki/XaddTool

[69]. University of California. (2014). Input and output templates. Recuperado el 1 de junio de 2015 de: https://boinc.berkeley.edu/trac/wiki/JobTemplates

[70]. University of California. (2014). Building BOINC applications on Linux. Recuperado el 1 de junio de 2015 de: https://boinc.berkeley.edu/trac/wiki/CompileAppLinux

[71]. University of California. (2014). The BOINC application programming interface (API). Recuperado el 1 de junio de 2015 de: https://boinc.berkeley.edu/trac/wiki/BasicApi

[72]. Open Source Notes. (2008). How to create your own work generator script Recuperado el 30 de mayo de 2015 de: http://blog.os-tools.net/en/how-to-create-your-own-work-generator-script-part-1/

[73]. University of California. (2014). Developing a custom validator. Recuperado el 1 de junio de 2015 de: https://boinc.berkeley.edu/trac/wiki/ValidationSimple

[74]. Leppänen, J. Serpent - A Monte Carlo Reactor Physics Burnup Calculation Code. (2015). Recuperado el 5 de septiembre de 2015 de: http://montecarlo.vtt.fi/

[75]. Leppänen, J. Serpent - A Monte Carlo Reactor Physics Burnup Calculation Code. (2015). User's manual. Recuperado el 5 de septiembre de 2015 de: http://montecarlo.vtt.fi/download/Serpent_manual.pdf

[76]. Becerra. J. Teorema del binomio. Recuperado el 10 de noviembre de 2016 de: http://132.248.164.227/publicaciones/docs/apuntes_matematicas/38.%20Teorema%20del%20Binomio.pdf

[77]. Eterovic, M. E, Monografía matemática. Recuperado el 10 de noviembre de 2016 de: http://www.fis.puc.cl/~fismat/El%20Trascendental%20N%C3%BAmero%20de%20Euler_2_.pdf

[78]. Goldberg, D. (1991). Apendix D. What Every Computer Scientist Should Know About Floating-Point Arithmetic. Recuperado el 10 de noviembre de 2016 de: https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

[79]. GNU MPFR. User's Manual Ed. 3.1.5. (--/09/2016). Recuperado el 15 de octubre de 2016 de: http://www.mpfr.org/mpfr-current/mpfr.pdf