Collaborative Grid Computing on RISC Architectures

Marc Muixi Mosquera
Máster en Ingeniería computacional y matemáticas (URV - UOC)

Félix Freitag

11/06/2017

Esta obra está sujeta a una licencia de
Reconocimiento 3.0 España de Creative Commons

FICHA DEL TRABAJO FINAL

Título del trabajo:	Collaborative Grid Computing on RISC Architectures
Nombre del autor:	Marc Muixí Mosquera
Nombre del consultor:	Félix Freitag
Fecha de entrega (mm/aaaa):	06/2017
Área del Trabajo Final:	Sistemas distribuidos
Titulación:	*Máster en ingeniería computacional y matemáticas*

Resumen del Trabajo (máximo 250 palabras):

En la actualidad es difícil imaginar un hogar sin dispositivos electrónicos conectados a internet. La gran mayoría de ellos pueden ser usados como instrumentos de trabajo si se es capaz de organizar una estructura que los pueda gestionar. Se puede pensar en estos dispositivos como componentes con poca potencia, pero a cambio estos son muchos y el propio progreso fomenta que año tras año se introduzcan mejoras en su hardware, ya se a nivel de capacidad de cómputo mejorando las CPUs (o GPUs los que dispongan de ellas), a nivel de incremento de capacidad de memoria así como el incremento de la duración de las fuentes de energía en caso que dependan de ellas.

La existencia de estos dispositivos y la mejora constante en los aspectos expuestos, da paso a imaginar un entorno donde estos dispositivos aprovechen sus tiempos ociosos para contribuir en cálculos científicos en bien de un interés común.

Abstract (in English, 250 words or less):

Nowadays, it is difficult to imagine our home places without internet connected electronic devices. Most of them can now be used as working instruments if we are able to organize an infrastructure to handle them. We can think on these devices as low performance components, however they are a big amout and the same progress encourages year after year to introduce hardware improvements, either at a level of performance capacity improving the CPUs (or the GPUs the devices that had them), either at a level of incrementing memory capacity as well as incrementing the duration of the source of energy for those devices depending on it.

The existence of these devices and the constantly improvement of the exposed fields, let us imagine an environment where these devices could take advantage of this situation by using their idle times to contribute in scientific calculations in a common interest.

Palabras clave (entre 4 y 8):

Grid - Volunteer - Android - RISC - Arm - Collaborative - BOINC - Architecture

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

La tecnología actual junto con la producción masiva y la economía de escala, ha permitido situar máquinas tremendamente potentes al alcance de la mayoría. Su utilidad así como la dependencia sistemática que tenemos de ellas nos ha llevado a que gran parte de la población, impulsada primordialmente por la necesidad de comunicación en largas distancias, disponga de algún dispositivo compuesto por una unidad de proceso, memoria y conexión a internet, capaz de realizar cómputos y ser programado, así como de comunicarse para obtener y transmitir información.

Algunos de los dispositivos que reúnen estas características son IOT gateways, miniPCs, Raspberry Pis, o los más comunes y disponibles, los smartphones. Estos últimos, a medida que pasan los años son cada vez más potentes y cada vez más accesibles económicamente hablando. De hecho, hoy en día los smartphones se han convertido en un imprescindible para todo usuario de internet y en concreto los terminales Android son cada vez más usados, más de 1.000 millones de terminales vendidos a finales de 2015 y aportan un hardware altamente capacitado a precios muy ajustados **[1]**.

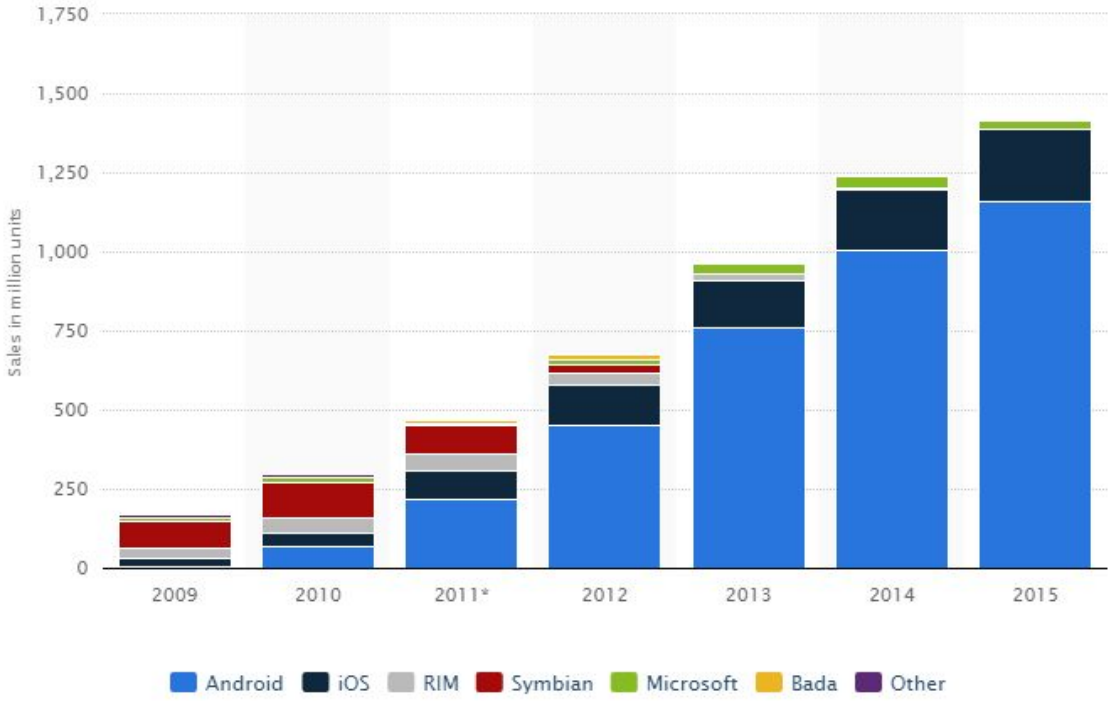

Figura [I]. Evolución ventas de smartphones por SO

Todos estos dispositivos de consumo personal representan una inmensa e imperecedera cantidad de poder de computación en las manos -literalmente- de cada individuo del planeta**[2]**. Estos recursos pueden ser aprovechados gracias a la computación grid, una configuración de la computación distribuida donde se aplica paralelismo sobre los nodos para resolver tascas con un propósito común, dividiendo el problema en tareas que puedan ser abordadas individualmente y mandando cada una de estas a los nodos disponibles.

1.2 Objetivos del Trabajo

- Revisión de la computación Grid y sus paradigmas.
- Análisi estructural de casos de éxito.
- Estudio del impacto de uso sobre el dispositivo.
- Exponer el uso de computación Grid sobre arquitecturas ARM.
- Poner en marcha un entorno Grid Volunteer Computing.
- Asignar trabajo a nodos Android en arquitecturas ARM.

1.3 Enfoque y método seguido

Este proyecto pretende mostrar la viabilidad de portar la computación Grid colaborativa sobre una red de dispositivos con procesadores basados en arquitecturas de instrucciones simples avanzadas ARM.
Con este trasfondo como objetivo, se busca revisar el estado del arte actual recorriendo las diversas soluciones existentes, categorizandolas e intentar una aproximación práctica a partir de las conclusiones extraídas.

El objetivo de seguir esta metodología es el de conocer y comprender cómo se están haciendo las cosas en el momento de este trabajo para poder aplicar las mejores técnicas en la aproximación práctica.

1.4 Planificación del Trabajo

Hitos:

- PEC 1: Planificar, organizar y definir el abasto del proyecto.
- PEC 2: Aproximación teórica.
- PEC 3: Aproximación práctica.
- PEC 4: Presentación de resultados.

TFM - Marc Muixi Mosquera Dec 15, 2016

Tasks 2

Name	Begin date	End date
PEC1 - Plan de trabajo	15/12/16	15/12/16
1 - Informe	8/12/16	14/12/16
0 - Gantt	8/12/16	14/12/16
PEC2 - Informe de progreso I	26/2/17	26/2/17
1 - Overview	16/12/16	27/12/16
3 - Revisión Estado del Arte actual	28/12/16	8/1/17
2 - Revisión soluciones existentes	9/1/17	20/1/17
4 - Especificación métricas	21/1/17	1/2/17
5 - Análisi impacto en nodo	2/2/17	13/2/17
6 - Hardware en cliente	14/2/17	25/2/17
PEC3 - Informe de progreso II	14/5/17	14/5/17
1 - Definición formal del sistema	26/2/17	9/3/17
2 - Implementación	10/3/17	20/4/17
3 - Iteraciones	21/4/17	8/5/17
Entrega final	11/6/17	13/6/17
1 - Conclusiones	14/5/17	12/6/17

Figura [II]. Planificación de tareas por fecha

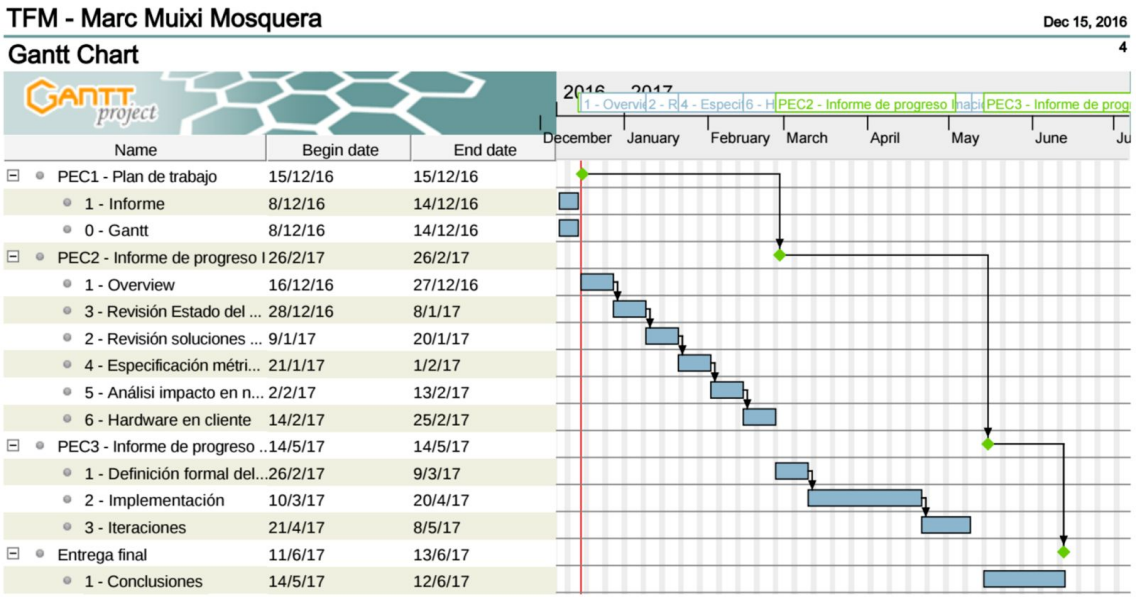

Figura [III]. Planificación de tareas Gantt

1.5 Breve sumario de productos obtenidos

- Tabla resumen de casos de éxito actuales de computación Volunteer Grid.
- Definición de capas de arquitectura de computación Grid
- Definición de métricas para medir los aspectos relacionados con el benchmarking de aplicaciones Grid.
- Conlcusiones acerca del uso de la computación colaborativa en dispositivos ARM.
- Capacidades de los nodos ARM para anexionarse a una red de computación colaborativa.

- Puesta en marcha de un entrono Volunteer Grid computing.

1.6 Breve descripción de los otros capítulos de la memoria

2. Overview:
Se define el paradigma Grid y se profundiza en sus orígenes y los motivos que vieron nacer esta estructura. Se dan ejemplos de uso y se explica el tipo de computación capaz de aprovechar el paradigma.

3. Revisión de las soluciones existentes:
A través de recorrer la soluciones de éxito conocidas, se genera una tabla comparativa para conocer cuáles son los modelos que consiguen dar el mejor rendimiento.

4. Revisión del Estado del Arte de la computación Grid:
Se muestra y se explica las bases teóricas que debe incorporar un sistema para unificar los nodos como Grid. Se especifican principios de computación distribuida, los modelos de datos y de ejecución, así como una taxonomía en función del tipo de computación.

A continuación se muestran las capas sobre las que basar una arquitectura Grid sin profundizar en los protocolos, dejando la libre elección a cada implementación particular.

Finalmente se muestra el flow de ejecución de una red Grid y con él los schedulers y el middleware así como otros temas relacionados.

5. Especificación de métricas para la medición sobre los nodos, la red y el modelo de interacción:
Se definen una serie de métricas para medir aspectos relacionados con el rendimiento y el consumo.

6. Análisis del impacto de uso sobre el nodo:
Se pretende dar una visión de las consecuencias del uso intensivo de la CPU que requiere la computación Grid sobre el nodo y cada uno de sus componentes.

7. Aproximación al hardware en nodo cliente:
Se define el paradigma Grid y se profundiza en sus orígenes y los motivos que vieron nacer esta estructura. Se dan ejemplos de uso y se explica el tipo de computación capaz de aprovechar el paradigma.

8. BOINC:
Se exponen sus características básicas, tanto de componentes como de uso.

2. Overview

2.1 Computación y redes Grid

El término *grid* nace en referencia a los grid de consumo eléctrico, donde el consumidor final que recibe la energía no sabe de dónde proviene ni como se ha generado. Aplicándolo a sistemas distribuidos, se entiende como la compartición de recursos interconectadas usando redes de interconexión de gran alcance junto a un middleware que actúa como homogenizador para hacer transparente el acceso a cada entidad.

Cuando se ha formado la red de nodos e interconexiones del grid, esta deviene un superordenador con componentes débilmente acoplados los cuales ofrecen una abstracción de un único computador. Para conectar estos nodos, se puede hacer uso de internet, de tal manera que se puedan aprovechar los commodity hardware de los usuarios particulares, los cuales pueden disponer de cualquier configuración tanto de SO, de recursos, de CPU,... pero que gracias a la acción de los middleware, estos equipos heterogéneos ofrecen acceso y uso a sus recursos sin importar las diferencias entre estos.

Los nodos trabajan de manera independiente y si no están plenamente dedicados a formar parte del grid, son ellos los que deciden el porcentaje de recursos que dedican a la red, de esta manera se respetan las políticas de uso locales. Esto define una característica muy importante de estos sistemas ya que en cualquier momento un nodo puede desaparecer o puede reducir/incrementar drásticamente el acceso a sus recursos.
Para gestionar los trabajos será necesaria un planificador de tareas, un grupo centralizado de nodos que gestionen el trabajo de los otros nodos.

2.2 Origen

El origen de la computación grid surge de la necesidad de compartir recursos dentro de la comunidad científica, la cual requiere computación de altas prestaciones. El principal desafío de la computación intensa local es el coste del hardware, electricidad, enfriamiento, espacio del suelo y de los administradores del sistema. Usando computación Grid, se pueden eliminar muchos de estos costes porque se actúa sobre recursos ya existentes. Por ejemplo, el coste de los nodos clientes (hardware y conexión a internet) no cuentan ya que el cliente ya los dispone para otros fines.
A primera vista, de cara al cliente que suministre el equipo, el único coste adicional a será la diferencia de consumo eléctrico de los sistemas cuando están uso frente a cuando se encuentran ociosos, sobre el cual se estudiará en este trabajo.

Debido a estas propiedades, es posible dividir y trasladar la responsabilidad del cómputo a diversas máquinas cada cual se responsabilizará de una parte. La división del trabajo en pequeñas tareas individuales es la clave para poder incrementar el poder de cómputo ya que, a diferencia de los clusters o los superordenadores, los nodos de las arquitecturas grid están poco acoplados, lo que se traduce en tiempos demasiado grandes para el paso de mensajes entre nodos.

2.3 Usos

Las aplicaciones más apropiadas para ser ejecutadas en un Grid son las de *coarse-grained,* aquellas donde la comunicación entre procesos tiene poca relevancia frente a la cantidad de computación necesaria y donde la potencia de paralelismo se sustenta en la cantidad de nodos que intervengan, siguiendo el paradigma de la computación de alto rendimiento High Throughput Performance (HTP).

Posibles aplicaciones de computación Grid pueden ser:

- Ciencias de la salud:
    - Descubrimiento de medicamentos. Desarrollo y testeo en bioinformática
- Energías:
    - Exploración de gas y crudo. Visualización de conjuntos de datos.
- Manufacturación:
    - Diseños de chips. Tests basados en simulación y evaluación.
- Finanzas:
    - Análisis de riesgos.
- Gubernamental:
    - Simulación y diseño. Coordinación de bases de datos distribuidas, utilidades en servicios.

2.4 Como aprovechar la computación Grid

Las aplicaciones actuales, al ser cada vez más profundas y proponer cada vez problemas computacionales más complejos, requieren una continua y creciente demanda de prestaciones y velocidad de computación.

Para dar solución a este problema, la primera aproximación se realiza sobre la mejora del hardware. Una de las mejoras más relevantes es el aumento de velocidad de reloj de la CPU gracias al aumento de transistores que se imprimen en estas junto debido a la disminución de su tamaño. Esta evolución sigue la “ley de Moore”, una ley empírica que anuncia lo siguiente: “la densidad

de circuitos en un chip se dobla cada 18 meses” (Gordon Moore, Chairman, Intel, 1965).

No obstante, este mismo avance tiene algunas propiedades contrarias a esta evolución, ya que a medida que aumenta la velocidad también lo hace la temperatura y los canales por donde circulan los electrones empiezan a dejar de ser fiables.

De esta manera, el aumento de potencia de computación debe buscar más vías, aquí es donde la computación Grid da respuesta a esta necesidad a través del paradigma de las aplicaciones HTC, el cual permite incrementar el número de tareas computacionales conjuntamente con el aumento de nodos encargados de estas. **[3]**

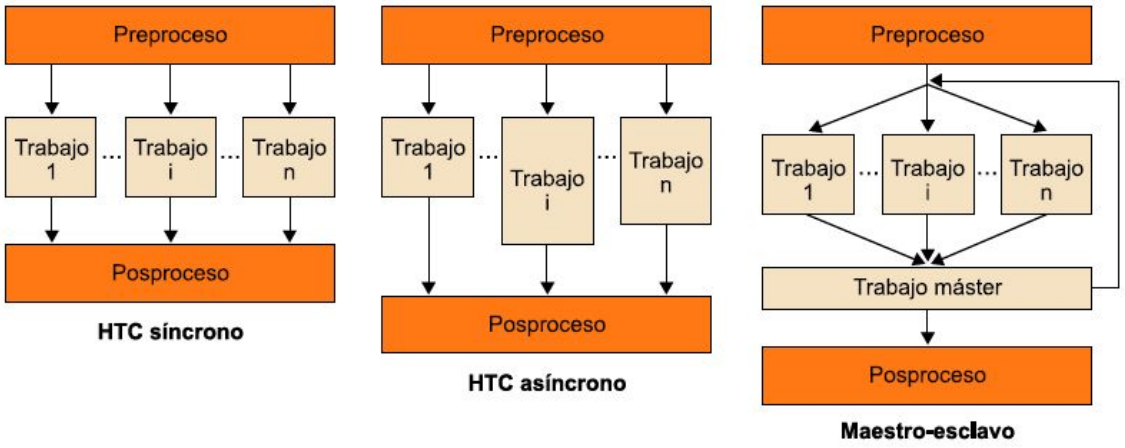

Figura [IV]. Esquema del paradigma HTC

3. Revisión de las soluciones existentes

Una buena manera de conocer una tecnología es recorrer varios de los sistemas que hacen uso de ella y comparar sus características. A medida que se van conociendo las soluciones actuales es posible desengranar las características que definen cada sistema y de esta manera compararlas para extraer conclusiones.

A continuación se presentarán casos de éxito donde se aplicó la computación Grid para resolver problemas de gran escala permitiendo la ejecución sobre nodos con arquitecturas ARM.

Mediante la revisión de las soluciones existentes, se busca encontrar las tendencias en la computación Grid así como las similitudes y diferencias entre ellas que indiquen hacia dónde se dirige este paradigma.

3.1 Análisi del mayor caso de éxito: "*Folding@Home"* (alrededor de los 100 petaFLOPS diarios)

- Volunteer Computing.
- Proyecto científico que analiza el comportamiento molecular de las proteínas al plegarse. El estudio busca la relación en la que dadas ciertas propiedades la proteína deja de plegarse para agregarse, lo cual está relacionado con enfermedades como el Alzheimer, el Parkinson o demás neuro-degenerativas.
- Operando a 5 PetaFLOPS constantes, ofrece más potencia computacional que la típica que puede conseguirse y operarse debido a los costes, espacio físico y carga eléctrica y de enfriamiento.El objetivo de la computación paralela no es el incrementar la velocidad de cálculo, sino lanzar simulaciones simultáneas sobre las que ir aproximando los modelos posteriores.
- Optimizado para el chip CELL de PS3, el cual es capaz de obtener una simulación de 500 ns por cada día de cálculo. Para obtener resultados significativos, es necesario del orden de milisegundos a segundos, que es cuando se producen los pliegues sobre las proteínas.
- Mejorar los códigos para los cores donde se ejecutarán (una pequeña mejora de rendimiento importa cuando se aplica sobre muchas máquinas a la vez)
- Su arquitectura está basada en un modelo *cliente-servidor*. Mediante un programa cliente preparado para correr bajo distintas arquitecturas y sistemas operativos descargado por el usuario. A partir de aquí se inicia el siguiente proceso de comunicación:
- Arquitectura en back **[2]**:

  1) se accede a un *assignment server* el cual le derivará a un *work server* concreto. Este actúa como un *scheduler* y un *load-balancer* global.

2) se accede a este *work server* el cual proporcionará un *work unit* para que el cliente lo trabaje en local.

3) dependiendo de la *work unit,* quizás se necesite una descarga extra desde un web server con contenido adicional necesario para el cómputo.

4) cuando el cliente termina el cómputo, se informa al *work server* asociado con los resultados. En caso que este no esté disponible, existen los *collection servers* que son capaces de recoger los resultados liberando de una posible sobrecarga al *work server* cuando volviese a estar en línea, además permite que el cliente continúe con otros cómputos.

5) finalmente hay un proceso interno de recogida de logs para generar estadísticas, muy necesario durante el proceso de recaptación y mantenimiento de usuarios.

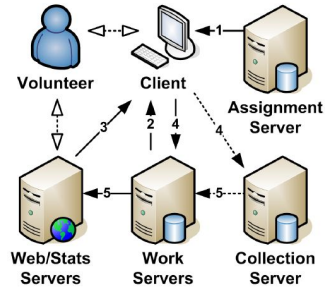

Figure 1. Folding@home architecture. Arrows depict data flow.

- Esta arquitectura puede ser ampliamente reutilizable para otros proyectos. Para ello sería necesario adaptar tanto el backend de los nodos servidor, para proporcionar, tratar, recoger y gestionar los diferentes datos del proyecto, así como el programa cliente, no tanto por sus protocolos sinó más por como este interactuará con el usuario para darle un feedback (de la misma manera que “Folding@home” hace con una representación 3D en movimiento de las proteínas estudiadas durante su uso en un sistema PS3)

- Capacidad de generar checkpoints de manera que se pueda reinicializar el cómputo a partir de estos. Habilidad para recuperarse después de caer.
- Se deben respetar las preferencias de configuración del volunteer (mala conexión, conexión a tramos ya que modem, evitar sobrecalentamiento de CPU, para cuando el portátil esté desenchufado....
- *Work Servers*: Generan las unidades de trabajo para un proyecto concreto, y son los encargados de aceptar, guardar, logear y analizar el trabajo completado. Cada nodo puede correr diferentes proyectos y es administrado por diferentes usuarios.
- *Assignment servers*: actuan de planificador global y de balanceadores de carga. Monitorizan los Work Servers y conocen su estado actual de trabajo, así como que CPU y OS montan. Se encarga de tomar la decisión (según carga de nodos y trabajo restante) de dirigir al cliente mediante un token para recoger una work unit.
- *Collection Servers*: Nodos de respaldo que se encarga de recoger los resultados del cliente en caso que el work node que los mandó no se encuentre disponible.
- El cuello de botella del sistema se encuentra en la gestión del backend. El mantenimiento de grandes clúster servers con un gran trato de almacenaje (cientos de terabytes) que gestionar, guardar y post-procesar.

3.2 Aplicaciones Grid sobre nodos ARM

Folding Home (2000) **[5]**:

- **PERFORMANCE (in TFlops):** 95.509 (2016)
- **MIDDLEWARE:** BOINC
- **CPU ARQ:** x86, x64, IBM Cell, ARM
- **CPU extras:** GROMACS (Assembly optimized code for each CPU)
- **GPGPU / GPU help:** Sí
- **GPU extras:** OpenMM
- **MULTIPLE PROJECTS:** no. Folding proteins simulation.
- **PLATFORMS SUPPORT:** Linux, macOS, Windows 64, Google Chrome, Android
- **REWARDS INCENTIVES:** Sí (logros personales)
- **ACADEMIC TARGET:** Sí

Distributed.net **[6]**:

- **PERFORMANCE (in TFlops):**. 30 (2001)
- **MIDDLEWARE:** Propietario (desconocido).
- **CPU ARQ:** AMD64, x86, IBM Cell, MIPS64, ARM, SPARC64
- **CPU extras:** OpenCL
- **GPGPU / GPU help:** Sí
- **GPU extras:** CUDA
- **MULTIPLE PROJECTS:** Sí. Criptography oriented
- **PLATFORMS SUPPORT:** Linux, BSD/OS, FreeBSD, Solaris/SunOS,macOS, MSDOS, Windows
- **REWARDS INCENTIVES:** -
- **ACADEMIC TARGET:** No

Samsung power sleep **[7]**:

- **PERFORMANCE (in TFlops):** -
- **MIDDLEWARE:** BOINC.
- **CPU ARQ:** ARM
- **CPU extras:** -
- **GPGPU / GPU help:** -
- **GPU extras:** -
- **MULTIPLE PROJECTS:** no. Similarity Matrix of Proteins (SIMAP)
- **PLATFORMS SUPPORT:** Android
- **REWARDS INCENTIVES:** -
- **ACADEMIC TARGET:** Sí

HTC app (power to give) **[8]** :

- **PERFORMANCE (in TFlops):** -
- **MIDDLEWARE:** BOINC.
- **CPU ARQ:** ARM
- **CPU extras:** -
- **GPGPU / GPU help:** -
- **GPU extras:** -
- **MULTIPLE PROJECTS:** yes. Limpiar el agua, lucha contra el cáncer, tratar schistosoma, arroz nutricional para el mundo.
- **PLATFORMS SUPPORT:** Android
- **REWARDS INCENTIVES:** -
- **ACADEMIC TARGET:** Sí

World Community Grid **[9]**:

- **PERFORMANCE (in TFlops):** 1100
- **MIDDLEWARE:** BOINC.
- **CPU ARQ:** x86, ARM
- **CPU extras:** -
- **GPGPU / GPU help:** -
- **GPU extras:** -
- **MULTIPLE PROJECTS:** sí. Smash Childhood Cancer, OpenZika, Help Stop TB, FightAIDS@Home, Outsmart Ebola Toghether, Mapping Cancer Markers
- **PLATFORMS SUPPORT:** Windows, Mac, Linux Ubuntu/Devian, Linux Fedora, Android
- **REWARDS INCENTIVES:** si (calificación por puntos, clasificación equipos, créditos proporcionales a FLOPS generados)
- **ACADEMIC TARGET:** Sí

Project / Attributes	**Performance (**	**Middleware**	**CPU architecture / CPU**	**GPGPU or GPU help**	**Multiple projects / names of**	**OS support**	**Rewards for incentive**

	in TFlops)		extras	admitted / GPU extras	projects		s
Folding @Home	95.509	BOINC	X86, x64, IBM Cell, ARM / GROMACS	Sí / OpenMM	No / Folding proteins simulation	Linux, macOS, Windows, Chrome, Android	Sí (logros personales)
distributed.net	>> 30	-	AMD64, x86, IBM Cell, MIPS64, SPARC64 / -	Sí / CUDA, ATI-Stream, OpenCL	Sí / enfocados a criptografía	Linux, BSD/OS, FreeBSD, SolarisSunOS, macOS, MSDOS, Windows	-
Samsung Power Sleep	-	BOINC	ARM / -	- /-	No / Similarity Matrix of Proteins (SIMAP)	Android	-
HTC Power to Give	-	BOINC	ARM / -	-/-	Sí / Limpiar el agua, lucha contra el cáncer, tratar schistosoma, arroz nutricional para el mundo	Android	-
World Community Grid	1100	BOINC	X86, ARM / -	-/-	Sí / Smash Childhood Cancer, OpenZika, FightAIDS @home, Mapping Cancer Markers	Windows, macOS, Linux Ubuntu/Debian, Linux Fedora, Android	Si (Clasificación por puntos, equipos. Creditos proporcionales a FLOPS calculado

Figura [V]. Comparativa de redes Grid con posibilidad de obtener los recursos de dispositivos con arquitecturas ARM

- **Performance**: evalúa el rendimiento medio del sistema.

- **Middleware**: framework usado para la abstracción sobre la red Grid.

- **CPU architecture**: arquitecturas sobre las que se puede ejecutar el sistema.

- **CPU extras**: mejoras o variaciones en de algún tipo en la CPU o en el código que esta ejecuta.

- **GPGPU or GPU help admitted**: indica si el sistema permite pasar cálculos a la GPU.

- **GPU extras**: arquitecturas o modelos de programación específicos que admite.

- **Multiple projects**: indica si admite diversos proyectos.

- **OS support**: indica los sistemas operativos para los cuales existe una implementación del cliente.

- **Rewards for incentives**: Indica que tipo de recompensas existen para incentivar al uso del sistema.

- **Academic target**: Indica si el uso computacional que se le da a la red Grid está destinada a la investigación académica.

  La tabla anterior, muestra los sistemas en uso propuestos y algunas de sus características. Todas ellas comparten la posibilidad de ser ejecutadas sobre nodos basados en arquitecturas ARM, en especial los sistemas de Samsung y HTC, los cuales están especialmente dedicados a estos en mayor parte porque son productos de asociaciones comerciales para campañas de marketing. Distributed.net, emprende campañas privadas y sus proyectos son de corta duración al conseguir solucionar los objetivos de las campañas.
  Finalmente, tanto World Community Grid como Folding@Home son ejemplos de un uso constante y duradero (ambos empezaron en la primera década del siglo), y se han ido adaptando con el tiempo a las nuevas tecnologías. Una prueba es el paso de ambos sistemas al uso de BOINC, un middleware que promete dar muy buenos resultados si nos fijamos en quienes lo usan. Un factor destacado de BOINC es que dispone de un cliente para Android.

  De esta tabla se extraen dos factores importantes:
  1. BOINC supone un buen middleware para crear redes de computación Grid sobre nodos ARM en un paradigma de Volunteer Computing.
  2. Los middleware tienen una gran importancia sobre las redes Grid.

3.3 Brokers de recursos y schedulers

Los siguientes paquetes nacieron como sistemas para gestionar jobs o tareas en plataformas de computación distribuida en local pero son aprovechados en las arquitecturas Grid para gestionar el trabajo y el acceso a los recursos :

Condor **[10]**:

- Paquete para la ejecución de trabajos por lotes sobre plataformas UNIX.
- La mayor ventaja es que permite una localización automática de recursos y alojamiento de jobs, check pointing y migración de procesos. Estas modificaciones no modifican el kernel UNIX, no obstante para usarlo se deben linkar sus librerías con el código fuente. Esto permite a Condor una monitorización de los recursos de los participantes, con los cuales genera una pool de recursos.

Portable Batch System (PBS) **[11]** :

- Se trata de un sistema de colas de lotes así como de gestión de carga de trabajo.
- Opera sobre máquinas UNIX.
- Permite establecer políticas propias para correr los jobs, tanto en tiempo como en espacio.
- Porvee un modelo extensible de autenticación y seguridad.
- Provee una GUI para la entrega de trabajos, traking y propósitos de gestión.

Sun Grid Engine (SGE) **[12]** :

- Los jobs esperan en una área de espera y las colas situadas en los servidores proveen a los servicios con los jobs.
- Se rige a través de la interacción del usuario, el cual solicita un job a la SGE y declara su perfil de requerimientos para el trabajo. Cuando la cola está lista para entregar un nuevo job, el SGE determina los jobs adecuados para esa cola y entrega el trabajo con la mayor prioridad y/o tiempo de espera en la cola.
- Intentará empezar nuevos jobs en la cola más adequada o con menos carga.

4. Revisión del Estado del Arte en computación Grid

El Estado del Arte en la computación Grid viene dado por los estándares de facto que actualmente rigen la mayoría y más importantes redes Grid que se encuentran en funcionamiento y que son claras muestras de casos de éxito.

Estos son fundamentalmente frameworks que incluyen todas las partes en el proceso de creación y gestión de una red Grid, dando soluciones a temas de planificación de tareas, seguridad tanto en el paso de mensajes como en la supervisión de resultados, gestión de organizaciones de uso, facilidad de uso así como definición de los protocolos de comunicación.

4.1 Líneas comunes

Heterogeneidad: la red debe estar preparada para componerse de una multiplicidad de dispositivos, arquitecturas, conjuntos de instrucciones y sistemas operativos diferentes.

Escalabilidad: la red debe estar preparada para el escalado y preparar la degradación de rendimiento a medida que la red crezca. En caso de necesidad de recursos para el cómputo, estos deben ser acercados geográficamente para mantener una latencia de trabajo adecuada.

Adaptación/Tolerancia a fallos: la red debe predisponerse para estar preparada a fallos.

4.2 Requerimientos para los datos y la infraestructura computacional

Administración jerárquica: es la manera que dispone la red para dividirse y abarcar el terreno global.

Servicios de comunicación: diversos servicios, desde comunicación confiable punto a punto a comunicación multicast. La infraestructura de comunicación necesita el soporte de protocolos para el transporte de volcado de información, streaming de datos, comunicación grupal, proveer a la red de una QoS como latencia, ancho de banda confiable, tolerancia a fallos y control del jitter.

Servicios de información: una red Grid es un entorno dinámico en el cual la localización y el tipo de servicios disponibles están constantemente cambiando. Es necesario disponer de un servicio para proveer al usuario de información.

Servicios de nombrado: como sistema distribuido, los nombres son usados para referir a máquinas, servicios y datos. Los servicios de nombrado proveen un espacio de nombres.

Distribución del sistema de archivos y caché: indispensable poder proveer al sistema distribuido de un espacio de nombres sobre el cual la entrega de los archivos esté proporcionada por una lógica de archivos distribuidos, replicados y cacheados para proporcionar la QoS esperada.

Seguridad y autorización: La red Grid, al ser una red distribuida hereda los aspectos de seguridad de estas y debe ser capaz de proveerlas cuando se requiera: confidencialidad, integridad, autenticación y responsabilidad.

Monitorización del estado del sistema así como tolerancia a fallos: para proveer un entorno robusto y confiable, es necesario poder monitorizar los procesos y extraer información de estos cuando sea necesario.

Manejo de recursos y calendarización: el núcleo de la red Grid debe conocer y actuar en consecuencia de los recursos disponibles, ya sea de tiempo de procesador, memoria, red, almacenamiento, o otros recursos. La finalidad de este control el la calendarización y asignación de recursos para optimizar los procesos.

GUI administrativa y de usuario: la red debe ofrecer una interfaz lo más intuitiva, sencilla y eficaz posible.

4.3 Modelos de datos Grid

Acceso ocasional a múltiples sistemas de almacenamiento terciarios: Puede ser necesario requerir acceso a metadatos y acceso uniforme a múltiples archivos de datos repositados en almacenamientos terciarios.

Análisi distribuido de conjuntos de datos masivos seguidos por catalogación y archivación: en muchas disciplinas científicas, una gran comunidad de usuarios requieren acceso remoto a grandes conjuntos de datos. Una técnica efectiva para mejorar las velocidades de acceso y reducir las cargas de red puede ser la replicación frecuente de los accesos a los conjuntos de datos en localizaciones elegidas para estar “cerca” de los usuarios eventuales. No obstante, organizar esta replicación para que sea a la vez confiable y eficiente puede suponer un problema. Los conjuntos de datos a ser movidos pueden ser grandes, por lo que aparecerán problemas de rendimiento de red y de tolerancia a fallos. Las localizaciones individuales donde se sitúen las réplicas pueden ofrecer características de rendimiento diferentes, en cual caso los usuarios (con herramientas de alto nivel) pueden ser capaces de descubrir

estas características y usar esta información para guiar la selección de réplicas .

Grandes conjuntos de datos de referencias: Una situación común es la necesidad de un conjunto entero de simulaciones que necesitan las mismas grandes referencias a conjuntos de datos. La gestión de estas bases de datos, que seguramente se localicen en un almacenamiento terciario, podrían ser manejadas por uno de los gestores de réplicas.

4.4 Modelos de computación Grid

Exportación de servicios existentes: el Grid provee un conjunto uniforme de servicios para exportar las capacidades de computación existentes, como los centros de supercomputadores a las comunidades existentes de usuarios. La idea es dar una entrada transparente de los servicios encapsulados en la red.

Procesos débilmente acoplados: colecciones de trabajos relacionados lógicamente que no tienen mucho en común durante su ejecución. Son trabajos que reciben un conjunto de datos como input a partir de los cuales generan otro output que se podrá relacionar con otro output generado por otro job.

Procesos distribuidos sobre pipelines/acoplados: Son procesos que esperan la salida de un proceso anterior para aceptarla como entrada propia y ejecutar su job.

Procesos fuertemente acoplados: Se trata de procesos que incorporan el paso de mensajes dinámicos durante su ejecución. El paso de mensajes es soportado por MPI (Message Passing Interface)

4.5 Servicios de gestión de alto nivel

Servicios de gestión de datos distribuidos: deben soportar el acceso y la manipulación de los datos distribuidos, tanto si se encuentran en bases de datos o en archivos. Estos servicios incluyen el acceso a bases de datos la traducción de datos, la gestión de réplicas, la localización de réplicas y las transacciones.

Servicios de workflow: soportan la ejecución coordinada de tareas de múltiples aplicaciones sobre múltiples recursos distribuidos Grid.

Auditación de servicios: deben soportar el grabado del uso de datos, asegurar el almacenamiento de estos datos, analizarlos con fines de fraude y detección de intrusión.

Instrumentación y monitorización de servicios: deben soportar el descubrimiento de “sensores” en el entorno distribuido, coleccionar y analizar la información de estos sensores y generar alertas cuando se detecten condiciones inusuales.

Servicios para determinar problemas sobre computación distribuida: estos deben incluir herramientas de tipo *dump, trace, log,...* mecanismos capaces de taggear eventos y correlacionarlos.

Servicios de mapeo de protocolos de seguridad: se debe permitir que los protocolos de seguridad sean mapeados transparentemente en los servicios de la plataforma nativa.

4.6 Taxonomía de aplicaciones de computación distribuida:

La computación Grid se caracteriza por dos modos de comportamiento, definidos por la relación entre los nodos dentro de la red:

- Enterprise, caracterizado por:
  - Enfocado a tareas computacionalmente largas dentro de una organización.
  - los nodos son conocidos y hay confianza en sus respuestas.
  - No hay necesidad para la replicación de resultados para comprobar su autenticidad.
  - No hay necesidad de entorno visual ni mostrar el avance de los resultados.

- Volunteer, caracterizado por:
  - Proyectos típicamente académicos.
  - Nodos desconocidos, aleatorios y fugaces.
  - Necesidad de replicación de resultados para comprobar su autenticidad estadística.
  - Sistemas de recompensa para favorecer el uso.
  - Necesidad de mostrar los resultados para mostrar eficiencia.

4.7 Organizaciones Virtuales

En las arquitecturas grid existen dos tipos de nodos, los proveedores y los consumidores.
Entre estos dos tipos, debe existir una manera de direccionar las disputas sociales y políticas para la obtención y compartición de recursos para el beneficio mutuo. Esto implica acuerdos de compartición de recursos entre proveedores y consumidores para un propósito común. Esta compartición de dispositivos, software, datos, y otros recursos está altamente controlado por los proveedores de recursos y consumidores definiendo claramente que se

comparte, a quien se le permite, y las condiciones bajo las que se produce la compartición. Un grupo lógico de unión de consumidores y proveedores es conocido como Organización Virtual (VO).

4.8 Arquitectura Grid

La arquitectura Grid viene definida por tres propiedades que toda aplicación Grid debería cumplir:

1) Debería proveer una coordinación de recursos descentralizada, teniendo en cuenta los problemas y situaciones de seguridad, politicas, pago y pertenencia.
2) Debería estar basada en estándares y protocolos abiertos para la autenticación, autorización, descubrimiento de recursos y acceso.
3) Debería proveer *Quality of Service* mediante el uso de recursos coordinados, proveyendo bajos tiempos de respuesta y un alto throughput para satisfacer las demandas del usuario.

El siguiente esquema **[13]** no provee una enumeración completa de todos los protocolos requeridos (servicios, APIs ni SDKs), sinó que identifica los requerimientos para clases generales de componentes. El resultado es extensible, una arquitectura abierta sobre la que se pueden implementar soluciones en función de los requerimientos de las Organizaciones Virtuales.

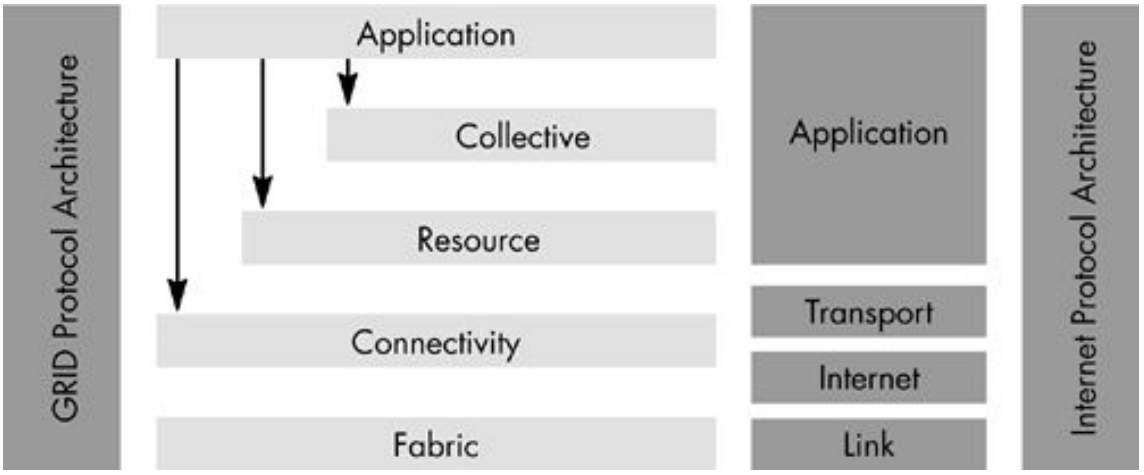

Figura [VI]: foster et al. la arquitectura de capas Grid y su relación con la arquitectura de Internet Protocol. Debido a que la arquitectura Internet Protocol extiende desde la red a la aplicación, existe un mapeo de las capas de Grid a las capas de internet.

Fabric: es la capa inferior y consiste en protocolos de recursos y conectividad, los cuales facilitaran la compartición de recursos individuales. Provee a los recursos de acceso compartido mediado por los protocolos de la red Grid.

Debe implementar mecanismos de investigación para descubrir los recursos que alberga su estructura, su estado, sus capacidades y los mecanismos de gestión de recursos para entregar *quality of service.*
El tipo de recursos que manipula esta capa pueden ser computacionales, de almacenamiento, de red, repositorios de código o bases de datos.

Connectivity: esta capa define la comunicación y los protocolos de autenticación para las transacciones de red. Su objetivo es proveer una comunicación sencilla y segura.
Los protocolos de comunicación incluyen internet (IP), transporte (TCP/UDP), aplicaciones (DNS y demás) y soportan espacios blancos para la inclusión de nuevos protocolos.
Los protocolos de autenticación deben ser capaces de proporcionar las siguientes funcionalidades:

- *Single sign on:* logeo una vez y acceso a todos los recursos de la capa de Fabric que permitan los permisos adquiridos.
- *Delegation:* se debe permitir a un programa correr en favor de un usuario si es capaz de acceder a los recursos sobre los que el usuario está autorizado.
- *Integritity:* debe proveer soluciones locales.
- *User-based trust relationships:* Los proveedores no están obligados a interactuar ante un usuario para acceder a los recursos de cualquiera de los proveedores.

Resource: Esta capa define protocolos para asegurar la negociación, inicialización, monitorización, control, cuentas y pago de operaciones compartidas en recursos individuales. Estos protocolos negocian con los recursos individuales e ignoran el estado global y se pueden entender como protocolos de información usados para obtener información sobre configuración, carga, políticas de uso y protocolos de gestión que negocien el acceso a recursos compartidos mediante el manejo de los requerimientos de recursos y operaciones a realizar. Los protocolos de gestión aseguran la consistencia de operaciones para un recurso compartido.

Collective: define protocolos y servicios globales de carácter general y captura interacciones a través de las colecciones de recursos. Se pueden encontrar entre otros:

- *Directory services:* para descubrir las propiedades de los recursos.
- *Coallocation, scheduling, and brokering services:* asignación de uno o más recursos para un propósito específico y sus tareas de calendarización.
- *Monitoring and diagnostics:* para fallos, detección de intrusos, sobrecarga y demás.

- *Data replication services:* para la gestión del almacenamiento con la finalidad de mejorar el rendimiento de acceso.
- *Grid-enabled programming system:* provee un modelo programatical para la búsqueda de recursos, seguridad, asignación y demás.
- *Workload management systems:* descripción y gestión de flujos de trabajo de componentes múltiples.
- *Software discovery services:* busca optimizar la selección de software.
- *Community authorization servers:* mejora las políticas comunes que gobiernan el acceso a recursos.
- *Accounting/payment and collaboration services.*

Applications: Esta capa incluye las aplicaciones de usuario que operan dentro de una Organización Virtual.
Las aplicaciones realizan llamadas a servicios definidos en cualquier capa -gestión de recursos, acceso a datos, búsqueda de recursos y demás- para realizar las operaciones deseadas. Estas dependen de APIs implementadas en SDKs los cuales llaman a protocolos Grid con servicios de red que proveen las capacidades al usuario final.

4.9 Grid entendido como un framework

Esto es una manera de enfocar la arquitectura Grid para aprovechar sus recursos bajo una economía de demanda. Se distinguen tres entidades que interaccionan con las demás para suplir sus necesidades:
Resources: están geográficamente distribuidos y son propiedad de diferentes individuos y/o organizaciones. Tienen sus propias políticas de acceso así como mecanismos para controlar los costes.

Resource owners: gestionan sus recursos usando sus schedulers locales y sus políticas específicas. Pueden rentabilizar el uso de sus recursos.

Grid users: estos consumen los recursos basados en las políticas definidas por los *resource owners.*

Esta arquitectura dirigida a la economia, potencia que todas las entidades obtengan un beneficio de la red. Como sistema de gestión de recursos utiliza los siguientes componentes**[14]**:

Meta Scheduler: también se les suele llamar *Grid Resource Broker,* proveen la capa entre los usuarios y los recursos vía los servicios del middleware.
Sus responsabilidades son la selección de recursos y el inicio de la computación. Además, puede proveer al usuario de una vista de la red Grid como una única entidad.

Grid Middleware: ofrecen los servicios que permiten al *Meta Scheduler* interacciona con los *local resource schedulers.* Entre los servicios que proveen, se encuentra la asignación de recursos, la transferencia de datos, la seguridad, la autorización y la autenticación de servicios.

Local Scheduler: son los responsables de ejecutar los programas o solicitudes del *grid users* en los recursos locales. Obedecen al conjunto predefinido de políticas impuesto por los *resource owners* y pueden ofrecer acceso a dispositivos locales, hardware y software.
Ejemplos son: *Portable Batch System (PBS), Sun Grid Engine (SGE) o Condor.*

User Applications: son el conjunto de programas a ser ejecutados por los *local schedulers* en recursos específicos.
Las aplicaciones pueden ser sequenciales si ejeutan un conjunto de instrucciones después de otro, paramétricas si requieren un conjunto de parámetros de entrada o paralelas si son capaces de correr instrucciones en paralelo.

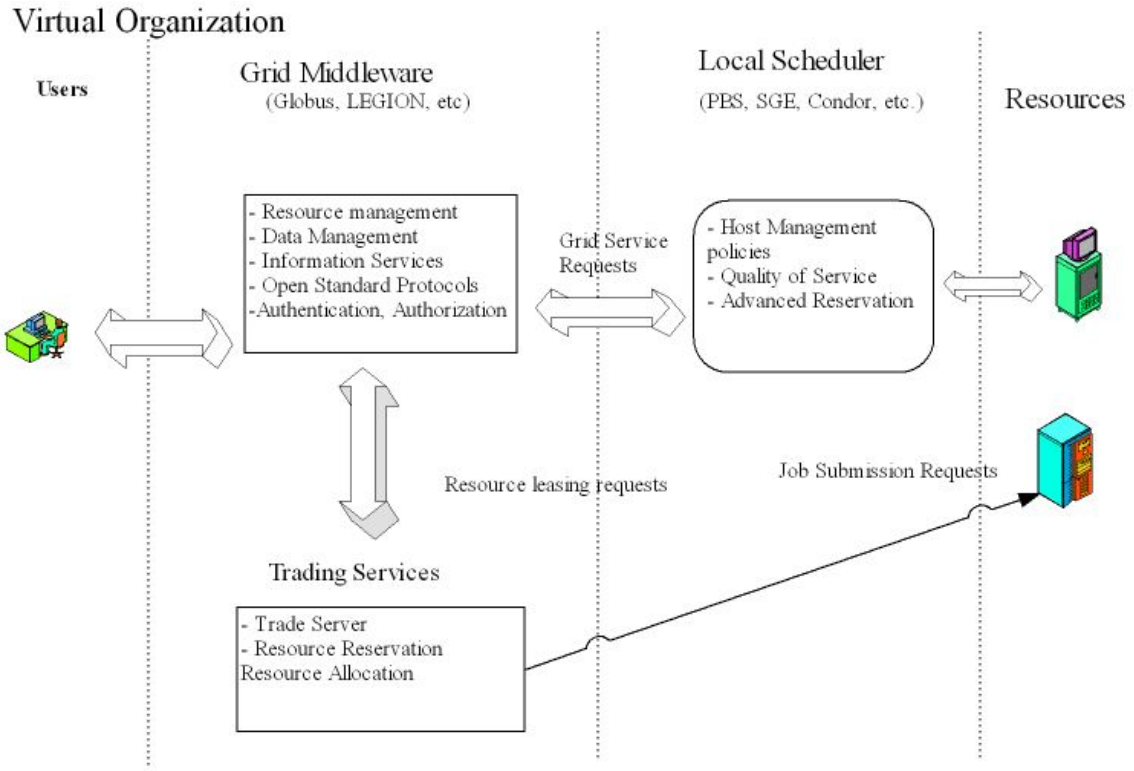

Figura [VII]. Componentes del sistema de gestión de recursos de una red grid dirigida a la economia

4.9.1 El middleware

El middleware se encarga de la gestión y control de la red, el software instalado en los recursos del grid (los nodos cliente), situado en una capa entre medio de su sistema operativo y la aplicación en cliente. Entre sus funciones están los esquemas de seguridad (autenticación de usuarios y recursos), que suelen implementarse en todos los middleware aunque pueden delegarse a los denominados brokers.
Otras de sus tareas es la replicación de los datos y metadatos, que requerirán que los nodos ejecuten una cierta tarea por cuestiones de eficiencia o tolerancia a fallos.

Proveen la infraestructura necesaria para soportar la ubiquidad, y la computación distribuida geográficamente. Estas herramientas de metacomputación proven tales servicios como computación de altas prestaciones, login singular a recursos distribuidos a través de múltiples organizaciones, así como APIs y protocolos para información, entrega de jobs y servicios de seguridad a través de las múltiples organizaciones. Proporcionan interfícies para esconder la complejidad de los sistemas que manejan por debajo.

4.9.2 Schedulers

Son una herramienta avanzada para la gestión de recursos usada en entornos heterogéneos de computación distribuida. Su objetivo es la optimización de los jobs que se encuentran a su responsabilidad y provee las siguientes funcionalidades:

- Acepta jobs por lotes
- Preserva y protege los jobs hasta que se ejecuten
- Ejecuta los jobs
- Entrega el output de vuelta a quien encargó el job

Estas herramientas utilizan complejas políticas de gestión y algoritmos sifisticados de caledarización para conseguir las siguientes funcionalidades:

- Collecta dinámica de datos de rendimiento
- Mejora de seguridad
- Políticas de administración de alto rendimiento
- Quality of Service (QoS)
- Tareas computacionales bajo demanda
- Distribución transparente a la carga asociada de trabajo
- Reserva de recursos avanzada
- Soporte para programas con check-pointing

4.9.3 Seguridad

Las aplicaciones Grid pueden querer autenticación, autorización, integridad y privacidad. En el contexto de los modelos de programación, esto acarrea ramificaciones.

Seguridad de tipo punto-a-punto se puede obtener mediante la integración de mecanismos de seguridad. Así mismo, para la verificación del código ejecutado en el cliente, este será enviado mediante una firma digital basada en criptografía de clave pública que asegurará el no repudio del generador del mensaje.

4.9.4 Tolerancia a fallos

La tolerancia a fallos está basada en la capacidad de migración de tareas. El middleware se encargará de manera transparente de generar checkpoints de manera regular de tal manera que pueda trasladarse el estado del cómputo y recuperarse en otro nodo.

4.10 Validaciones en Volunteer

En el paradigma de la computación volunteer, no se puede confiar en los resultados recibidos de todos los nodos. Aunque la gran mayoría de resultados sean de confianza, hay que asegurar que los resultados se han obtenido mediante la ejecución del trabajo y no han sido corrompidos por errores de memoria, errores en disco, errores en CPU o viruses. Esto significa que es necesaria una validación para asegurar que los resultados representan una respuesta correcta. **[15]**

4.11 Paso de mensajes

Message Passing Interface (MPI) **[16]**es un estándar para pasar mensajes entre programas provisto de portabilidad y facilidad de uso con un conjunto de base de rutinas bien definido.

MPI representa el concepto básico de comunicación entre procesos mediante mensajes.

El paso de mensajes es un paradigma usado en la mayoría de sistemas paralelos, especialmente aquellos de memoria distribuida.
Se caracteriza por:

- Provee una comunicación eficiente y una API para evitar copiado de memoria-a-memoria y permitir la superposición de la computación y la comunicación.
- Puede ser usado en entornos heterogéneos.
- Provee una interfície de comunicación confiable: el usuario no necesita tratar con los errores de comunicación. Estos son delegados en el sistema de comunicación que va por debajo.
- MPI es portable y puede ser implementado en gran diversidad de arquitecturas sin cambios significativos.
- La interfície de MPI es un lenguaje independiente y diseñado bajo la premisa de ser *thread safety.* MPI provee diversas funcionalidades que pueden mejorar el rendimiento o las computadores paralelas escalables mediante interprocesos especializados de comunicación hardware.

5. Especificación de métricas para la medición sobre los nodos, la red y el modelo de interacción

Al traspasar el cómputo a los nodos productores, estos participan dando a la red Grid el tipo de recurso para los que han sido adheridos a esta, pero a cambio sufren de un desgaste que corre a su cuenta. Es necesario conocer y cuantificar este desgaste para conocer la viabilidad de la red entendida como conjunto, así como el gasto directo e indirecto que supone para los nodos productores.

Para medir estos aspectos se hace uso de métricas asociadas a los procesos o componentes que participan en la computación Grid.

5.1 Medidas de rendimiento

- **Floating-Point Operations per Second (FLOPS)**: mide el rendimiento de cómputo.

$$FLOPS = sockets * \frac{cores}{socket} * clock * \frac{FLOPs}{cycle}$$

- **Instructions per Second (IPS)**: mide la cantidad de instrucciones por segundo que es capaz de ejecutar una CPU.

$$IPS = sockets * \frac{cores}{socket} * clock * \frac{IPs}{cycle}$$

- **GigaHertz (GHz)**: indicativo de la velocidad de una CPU.

- **Benchmarking**: indicativo de rendimiento de un sistema.

5.2 Medidas energéticas

- **Potencia eléctrica (Watt)**: cantidad de energía convertida, utilizada o disipada en un segundo.

- **Amperio - Hora (Ah):** Cantidad de carga eléctrica que circula por una sección de un conductor por donde pasa una corriente de 1 Amperio durante 1 hora.

- **FLOPS per watt (Performance per watt)**: número de FLOPS entregados dada la poténcia de 1 Watt.

- **CPU consumption per GHZ (Watt/GHz)**: velocidad conseguida dada la potencia de 1 Watt.

- **CPU MIPS efectivos por MHz(Mips/MHz)**: Número de instrucciones ejecutadas por 1 MegaHerz.

5.3 Medidas de consumo

- **Ciclos de carga:** consumo total de la carga completa de una batería.

- **CPU speed on idle**: velocidad de la CPU en estado de reposo.

- **CPU speed during full usage**: velocidad máxima de la CPU la cual es entregada para mejorar el rendimiento durante los cálculos.

5.4 Medidas económicas

- **Coste TFLOPS:** valor de mercado de 1 TFLOPS.

- **Coste TFLOPS anual:** valor de mercado de mantener 1 TFLOPS durante 1 año.

5.5 Medidas de red (transferencia de información)

- **Latencia**: tiempo transcurrido entre el momento en el que el origen empieza a transmitir los datos y el momento en el que el destino empieza a recibir el primer byte.

- **Ancho de banda**: ratio de recepción de datos en destino después de que haya empezado a recibir el primer byte.

El tiempo de transmisión de datos vendrá dado por:

$$t = l + \frac{n}{b}$$, con

tiempo de transmisión: t
latencia: l
Número de bytes de la transmisión: n
Ancho de banda del canal: b

6. Análisi del impacto de uso sobre los nodos

El coste de aportar ciclos de CPU/GPU a primera vista puede parecer que sólo se desglosa en el consumo eléctrico necesario durante el tiempo de cómputo. No obstante, existen una serie de efectos colaterales que pueden influir en el coste de esta computación, debido a consecuencias indirectas del uso de la CPU/GPU.

En este apartado se pretende comprobar si estos efectos colaterales son suficientemente notorios como para tenerlos en cuenta sobre el coste del cómputo en el nodo.

6.1 Calentamiento CPU/Dispositivo

Durante la ejecución del cálculo los dispositivos tienden a calentarse debido al uso intensivo de sus unidades de cálculo, las cuales se suelen poner a mayores velocidades para incrementar el rendimiento. Este calentamiento se debe controlar para salvaguardar al dispositivo, el cual queda expuesto y puede sufrir graves desperfectos.

En el caso de las baterías, el calentamiento puede afectar directamente en la velocidad de degradación de estas ya que, necesitan encontrarse dentro de un rango de calor y humedad específico para mantener la carga en estado óptimo. Se calcula que por cada 10 grados centígrados de aumento de la temperatura, el ratio de autodescarga se dobla.

Si durante la ejecución del cómputo Grid el calor que genera la CPU es excesivo y no se disipa bien, la batería se sobrecalentará lo que se traducirá en pérdida de capacidad con el tiempo.

El kernel del SO se debe encargar que sobrepasado cierto umbral, normalmente sobre los 50º, el perfil del dispositivo se transforme a un perfil de carácter ahorrador donde se reduce el rendimiento con el fin de enfriarlo.

En caso de equipos de sobremesa se puede hacer uso del enfriamiento de diversos tipos, como la refrigeración líquida sobre cada uno de los componentes ya que se encuentran separados y hay espacio para insertar estos sistemas de refrigeración. No obstante, en el caso de los smartphones, donde junto al SoC se encuentran todos los demás componentes y sin apenas refrigeración, controlar el pico de temperatura se convierte en una característica esencial para salvaguardar la continuidad de los componentes.

6.2 Consumo eléctrico

El consumo eléctrico derivado de la computación Grid vendrá dado por el tiempo de computación proporcionado a la red. El consumo de energía en estos intervalos supone un aumento derivado del incremento de potencia que se debe entregar a las CPU/GPU. Las arquitecturas ARM se caracterizan por su reducido consumo, en concreto para arquitecturas Cortex tipo A (las usadas en smartphones) siendo unos valores aproximados de consumo al alza en un momento de máximo estrés alrededor de los 2 Watts por cada clúster de CPU, 2 Watts más destinados a la GPU y quizás 0.5 Watts para la MMU y el resto del SoC**[17][18][19]**.

Aunque el consumo es muy reducido, tiene un impacto demasiado elevado sobre las actuales baterías que rondan los 3000 mAh las cuales podrían aguantar alrededor de 6-7 horas este consumo. Es por ello que las actuales aplicaciones se ponen en marcha al conectar los dispositivos a fuentes de alimentación y no cargar a las baterías con este uso de manera prolongada.

No obstante, el total de esta configuración no llega a los 5 Watts, la cual cabe comparar con un procesador CISC de alto rendimiento, por ejemplo un i7, que necesita de más de 150 Watts durante un uso intensivo de sus núcleos.

6.3 Desgaste de batería

Las arquitecturas ARM pueden encontrarse en gran medida implementadas sobre dispositivos que obtienen la energía mediante baterías.

Actualmente el almacenamiento de energía en baterías se compone de elementos químicos enlazados mediante un electrolito y basados en una reacción de oxidación-reducción en ánodo y cátodo respectivamente. Esta solución responde a las necesidad actuales pero su uso no es ilimitado. Dependiendo del tipo de cuba electrolítica principal de la celda electro-química que compone la batería (las usadas en los smartphones usan iones de litio), esta dispone de un número de ciclos de descarga que una vez superados, esta empezará a perder capacidad de carga así como potencial.

El desgaste de batería durante la computación Grid será alto debido a la necesidad de voltaje de las CPU para poder incrementar su rendimiento. Más allá del consumo eléctrico visto en el punto anterior, hay que tener en cuenta el consumo de ciclos de vida de la batería derivados del tiempo de computación Grid. Esto conlleva un gasto directo sobre el desgaste del tiempo de vida de la batería y aunque en algunos dispositivos es posible cambiarla, muchos traen las baterías incrustadas y no es posible el reemplazo, por lo que la computación Grid sobre estos tendría un impacto directo sobre el tiempo de vida del dispositivo.

6.4 Desgaste de otros componentes del nodo

Las memorias de smartphones, así como IoT Gateways o Development boards como las Raspberry Pi se basan en memorias de tipo eMMC (embedded Multi-Media Controller). Estas se caracterizan por su bajo consumo, su rápida velocidad de lectura/escritura y por su modo de almacenamiento basado en memorias FLASH **[20][21]**. Este modo de almacenamiento es muy adecuado para el tipo de dispositivo sobre el que irá montado una CPU ARM, no obstante uno de sus principales inconvenientes es que dispone de un número limitado de ciclos de programación - borrado (P/E cycles **[22]**).

Para evitar la rápida degradación que supondría el borrado continuo sobre los mismos bloques de memoria, estos se acompañan de mecanismo que invalida los bloques a borrar y escribe en nuevos bloques, de tal manera que el borrado se produzca sólo en momentos cuando no hay más almacenamiento disponible.

La computación Volunteer se basa en la descarga continua de nuevos paquetes de datos que se almacenan en memoria. Este proceso ejecutado de manera continua puede promover ciclos de programación - borrado lo que se traduce en una reducción de la vida de las memorias.

6.5 Impacto directo en el tiempo de vida del nodo

El impacto directo sobre el tiempo de vida del nodo se focaliza principalmente en los componentes de energía y memória. En algunos dispositivos ambos pueden ser externalizados, la energía se la entrega una fuente de alimentación externa y la memoria es almacenada en tarjetas externas de fácil intercambio.

No obstante, en dispositivos como los smartphones pueden nos ser fácilmente intercambiables, por lo que el desgaste/deterioro producido por el uso de la red Grid sobre estos puede será proporcional al tiempo de vida de estos dos componentes.

6.6 Impacto indirecto sobre el uso en el usuario

La ejecución de la computación Grid está pensada para realizarse siempre siguiendo las políticas locales, por lo que el cliente será capaz de elegir los modos de ejecución según sus preferencias.

Por defecto estos traen configuraciones que activan el cálculo sólo al cumplir ciertas condiciones que aseguran que el impacto será menor, en caso de BOINC son:

- Batería superior al 90%.
- Dispositivo conectado a fuente de alimentación.

- Limitación sobre el número de núcleos a usar (dejar al menos un núcleo libre).
- Restricción sobre el uso de RAM y memoria dependiendo del dispositivo.
- Modo de uso: screenSaver o pantalla apagada.
- Modo de conexión: sobre una red WIFI.

6.7 Consumo de datos / tráfico derivado

Por defecto los clientes configuran que el intercambio de datos se produzca siempre sobre WIFI, no obstante el cliente es quien finalmente decide.

El consumo de datos viene en mayor medida producido por la descarga de contenido y subida de resultados.

6.8 Comparación monetaria

El impacto monetario de para el usuario que proporciona un nodo a la red de computación Grid se desprende de tres factores:

- El consumo de datos: Al poder realizarse sobre redes WIFI, este consumo es opcional y depende de las tarifas de cada operadora.
- El consumo de energía: Según datos del Lawrence Berkley National Laboratory **[23]**, son necesarios 3.68 W para cargar un smartphone (siendo esta potencia la de las mayores entre los dispositivos que usan CPUs ARM).

Sobre este consumo, se podría simular un uso de Computación Grid de 8 horas diarias (coincidiendo con las horas de sueño por ejemplo para no desviar recursos durante el uso del usuario) durante un año y serían necesarios 21’52 kWh para cargar suplir de energía al dispositivo durante este tiempo.
En España, aproximando el precio del kWh sobre los 0’13€ **[24]**, tendríamos que por un año el coste de dejar nuestro dispositivo realizando computación Grid durante 8 horas al día estaría alrededor de los 2,8 € al año, un coste más que asumible.

- El desgaste del dispositivo: Como se ha expuesto, podemos encontrar el desgaste sobre la batería y sobre la memoria interna. La primera puede ser reemplazada en algunos casos, mientras que la segunda debe ser configurada para usar tarjetas de memoria en lugar de la memoria interna, las cuales pueden ser también reemplazadas.

Sobre las baterías, son el componente más expuesto y con mayor degradación.

6.9 Impacto ecológico

La mayoría de dispositivos electrónicos dedicados al consumo se caracterizan por la producción en masa capaz de reducir los costes de producción y permitir la adquisición masificada de estos.

Este proceso pone en evidencia dos efectos colaterales que la computación Grid podría equilibrar:

- Energía de creación / energía consumida: La cantidad de energía que un dispositivo consume durante su tiempo de vida es negligible comparada con la energía necesaria para ponerlo en producción.

En este sentido aprovechar estos recursos para realizar cálculos contribuye a amortizar la producción del dispositivo en términos de eficiencia ecológica.

- Tiempo de vida de los componentes: La facilidad de adquirir nuevos componentes más potentes a precios más reducidos, produce que a escala individual se elija la adquisición de un nuevo dispositivo en un tiempo mucho más reducido del tiempo que sus componentes necesitan para empezar a desgastarse.

En este sentido, el uso intensivo de estos componentes ayudará a equilibrar el ciclo de vida de tal manera que en el momento de desecharlos, sus capacidades estarán más cerca de la muerte funcional del dispositivo, lo que justifica en mayor grado su cambio.

7. Aproximación al hardware en nodo cliente

Los nodos clientes son todas aquellas máquinas dentro de la red Grid que albergan los recursos. Este trabajo se centra sobre los nodos basados en arquitecturas sobre CPUs ARM debido a su extensa implantación, por lo que la aproximación estará enfocada a este tipo de dispositivos, analizando sus características para este tipo de computación así como las maneras para anexionarlos a una red Grid.

7.1 Arquitecturas ARM

Las arquitecturas ARM tienen un enfoque de diseño basado en RISC (Reduced Instruction Set Computing), lo que físicamente implica que el chip tendrá menos transistores. La principal ventaja de esta configuración reside en la reducción de costes de fabricación, de calor y de energía, propiedades muy deseables para dispositivos portables que funcionen con baterías.

Otras propiedades importantes que fundamentan esta arquitectura son:

- Son más lentas: para conseguir un menor consumo de operaciones se usan transistores de poca velocidad, los cuales ofrecen una mejora de las fugas de corriente así como del voltaje mínimo.

- Son más pequeñas: se usan menos transistores, sobretodo debido al la arquitectura basada en el juego reducido de instrucciones RISC. Esto implica que las operaciones costosas son procesadas como pequeñas partes, a la expensa de más código máquina (algo que es transparente para el programador al hacer uso de compiladores). Esto significa que ARM tiene menos partes de un sólo uso que estén absorbiendo energía mientras no son usadas, así como que son más pequeñas y menos costosas.

Se puede observar la diferencia de configuración desde la siguiente ecuación:

$$\frac{time}{program} = \frac{time}{cycle} * \frac{cycles}{instruction} * \frac{instructions}{program}$$

Mientras que CISC (Complex Instruction Set Computing) minimiza el número de instrucciones por programa sacrificando el número de ciclos por instrucción, RISC hace lo contrario, reduce los ciclos por instrucción al coste de aumentar las instrucciones por programa**[25]**.

- Uso dinámico de los núcleos: las nuevas configuraciones ARM permiten disponer de varios núcleos con diferentes configuraciones, con diferentes consumos para diferentes usos. Mediante el mecanismo de *Heterogeneous Multi Processing (HMP)* ***[26]***, se decide cuantos y cuales núcleos usar en cada momento en función de los requisitos del programa en ejecución. Este mecanismo permite tanto el mínimo de potencia, mediante el apagado de los núcleos más potentes, así como la máxima entrega de potencia con todos los núcleos disponibles encendidos y a máxima potencia.

Para modificar el uso de potencia se usan los mecanismos de *Dynamic Scale Voltage* que se encargan de entregar voltaje en función del rendimiento que sea necesario para el núcleo **[27]**.

Estas propiedades hacen que las arquitecturas ARM sean la opción más elegida para construir los núcleos y el conjunto de instrucciones de las CPU por parte de los principales actores en el mercado actual, siendo estos empleados en la mayoría de dispositivos Android, con más del 95% de cuota de mercado en procesadores para smartphones, los cuales ascienden a más de 3.000 millones de unidades en todo el mundo**[28]**.

La mayoría de sistemas operativos pueden funcionar sobre esta arquitectura, con mención especial a los LTS kernels de Linux sobre los que se basan las distribuciones de Android.

Hoy en día, existe una grandísima variedad de dispositivos que disponen de una CPU basada en ARM, entre ellos podemos destacar algunos que podrían aprovechar su potencial para unirse a redes Grid**[29]**:

- Tablets
- Smartphones
- IoT
- Set-top-boxes
- SmartTv
- VideoGames
- Robots
- Servers
- Development boards

7.2 Requisitos para la ejecución Grid sobre ARM

Para poder formar parte de la red de computación Grid, los dispositivos conectados a ella (recursos) deberán ser capaces de soportar la carga de la

computación. Mediante el estudio de las soluciones actuales veremos por donde se mueven estos requisitos mínimos:

Requerimientos técnicos mínimos en HTC Power To Give **[8]**:

- Android > 4.4 (KitKat) **[30]**
- CPU: 1.5Ghz dual-core, 1 Ghz quad-core
- 1GB RAM
- Alimentación eléctrica conectada
- Conexión a internet
- Battería superior al 90%
- Máxima temperatura alcanzada por la batería durante su uso fijada en 40º

Requerimientos técnicos mínimos en Folding@Home **[31]**:

- Android > 4.4

Requerimientos técnicos mínimos en World Community Grid **[32]**:

- Android > 4.1 **[33]**

Requerimientos técnicos mínimos en BOINC **[34]**:

- Android > 2.3 **[35]**

Requerimientos técnicos mínomos para en Distributed.net **[36]**:

- Requiere soporte para uso de eabi **[37]**

Así como *HTC Power to Give* define muy claramente unos requisitos mínimos que el dispositivo debe disponer para su ejecución, nos encontramos por otro lado que los clientes basados en BOINC (*Folding@Home y World Community Grid)* tan sólo definen una versión de Android. No obstante, a partir de la versión de Android se puede extraer que el mínimo de RAM permitido para dispositivos que monten esta versión son 320 MB aunque la mayoría esté por sobre los 512 MB.

A partir de la arquitectura ARMv7 los conjuntos de instrucciones ya disponen de llamadas a los módulos de *VFP* **[38]** para operaciones de coma flotante, así como el módulo *NEON* para instrucciones del tipo Single Instruction Multiple Data (SIMD) **[39]**.

Estos requisitos ya están sobrepasados por más de un 60% de los dispositivos Android (sumando todos los dispositivos superiores a Android 5.0 Lollipop **[40]**), ya que a partir de Lollipop, Android ofrece soporte para la arquitectura de ARMv8 con soporte para 64 bits además de una mejora en sus GPUs**[41]** .

Revisando estas características, parece que las limitaciones vienen dadas por un mínimo de RAM así como por unas características básicas de software que

proporciona una versión mínima. No así unas especificaciones claras de componentes necesarios o capacidad de CPU como se hace desde *HTC Power To Give.*

Una manera que podría tener la red Grid de comprobar el rendimiento de los dispositivos de recursos sin tener que fijar versiones mínimas en sus clientes, podría pasar por una regulación inteligente de la propia red al planificar los trabajos y repartirlos entre los recursos. Mediante métricas de tipo temporal o incluso pequeños benchmarks se podría tantear a los nodos de manera que la red fuese capaz de elegir en función de sus necesidades y de sus recursos.

7.3 Soluciones para adicionarse a una red Grid

Actualmente los proyectos Volunteer Computing ofrecen clientes que pueden ser descargados y ejecutados que simplifican la adición a la red. Estos son distribuidos como clientes nativos escritos para un SO concreto, en forma de aplicación para dispositivos basados en Android o pueden ser empaquetados en Dockers.

8. BOINC

Hoy en día, todos los grandes proyectos de computación Volunteer Grid hacen uso de este sistema como se ha visto en el capítulo 3.Revisión de las soluciones existente. Proyectos como Einstein@Home **[42]**, World Community Grid**[8]** o SETI@Home**[43]**, incluyen BOINC como parte de su ecosistema, delegándole las tareas de middleware.

El hecho que estos proyectos confiaran en esta plataforma, ha impulsado la investigación de esta plataforma para conocerla más a fondo.

8.1 Volunteer Computing con BOINC

BOINC **[44]** es una plataforma de software desarrollada en la universidad de Berkeley para la computación Volunteer Grid. Esta plataforma se conforma de clientes para las diferentes arquitecturas, junto con la parte de servidor, componentes web y APIs para conectar con otros componentes.

Para ello, BOINC lidia con los siguientes aspectos de la computación Volunteer Grid:

- Computación distribuida
- Desarrollo de aplicaciones para usar este paradigma
- Creación de proyectos

Mediante estas herramientas que ahora se presentarán con más detalle, este sistema permite crear un servidor BOINC donde poder alojar proyectos de computación Volunteer. El servidor se encargará de gestionar los jobs para su entrega y comprobación, así como demás tareas heredadas de la comunicación cliente-servidor segura, como autenticación, validación etc.

Por otra parte, se ofrece el cliente BOINC, un conjunto de librerías que pueden usarse como cliente final o como pareja a una aplicación propia que haga uso de la infraestructura de BOINC.

Al ser un proyecto con finalidades científicas y con una parte desarrollada por la comunidad, este es un software libre que en que su licencia permite la copia, distribución y modificación bajo los términos de GNU Lesser General public License desde la versión 3, por lo que su código fuente es abierto, está repositado en GitHub y se puede usar para cualquier finalidad.

8.2 Conceptos básicos

Proyecto: Es la entidad independiente que realiza la computación distribuida. Se compone de sus aplicaciones, bases de datos, páginas web y servidores de tal manera que queda aislado.

Cada proyecto se identifica por su *master URL,* la dirección de su sitio web.

Aplicación: incluye diversos programas, para diferentes plataformas y un conjunto de *workunits* y *results.*

Plataforma: es el objetivo de compilación, formado por el conjunto arquitectura de CPU más sistema operativo.

Versiones de aplicación: Cada aplicación se puede segmentar en una secuencia de versiones, donde cada una es una compilación particular para una plataforma concreta.

Workunit: es la tarea computacional que desea realizarse, también conocida como *job.* Puede incluir archivos de entrada y puede definir atributos como los requerimientos de ejecución en el nodo ejecutor, así como fechas deadline de entrega de resultados.
Los *workunits* se asocian con la aplicación, no con las versiones de aplicación, lo que significa que definen el cómo pero no el qué.

Resultado: describe la instancia de computación con su estado, ya sea por empezar, en progreso o completada. Cada resultado está asociado directamente a un *workunit* y en algunos casos pueden existir varias debido a las réplicas del *workunit.*

Cuenta de usuario: Cada usuario dispone de una cuenta que le permite autenticarse. Gracias a esto, se provee un programa de recompensas basadas en *crédito* que se consigue según el rendimiento de computación del nodo.

Crédito: recompensa que se entrega al usuario por el rendimiento de su máquina. Este rendimiento se mide en *Cobblestone* el cual 1 unidad (o un crédito) equivale a 1/200 del tiempo de CPU de una máquina que ha rendido a 1 GigaFLOP constante (basado en el benchmark de *Whetstone* **[45]**) durante 1 día.

8.3 Computación distribuida

8.3.1 Modelo computacional

- Plataformas: el cliente de BOINC está precompilado para ser ejecutado en varias plataformas. Por cada una de estas plataformas, es posible usar la red Volunteer Grid provista por BOINC **[Annexo 1]**.
- Jobs: cada job está computesto por un *workunit* y uno o más resultados.
- Errores y redundancia: por cada *workunit,* se provee una forma de computación redundante en la cual cada computación se realiza en múltiples clientes y se

comparan los resultados, siendo estos aceptados cuando hay un consenso entre todos los resultados recibidos.
- Distribución del trabajo: mediante una política de distribución de trabajos, se procura mantener a los nodos lo máximo ocupados posible.
- Calendarización local: es un procedimiento que procura reducir el tráfico de red. Se procura mandar unidades de trabajo que necesitan la misma entrada a los nodos que ya disponen de estos archivos.
- Goteo de mensajes: la comunicación está preparada para permitir el envió de mensajes del cliente al servidor durante la ejecución de una tarea.

8.3.2 Modelo de datos

- Almacenamiento: el modelo de almacenamiento está basado en ficheros. Los principales ficheros a destacar son por un lado los xml de entrada y salida de los jobs, y por otro todo el conjunto de componentes de las aplicaciones: ejecutables, librerías, etc.
- Compresión: los archivos mandados y recibidos son comprimidos en gzip.

8.4 Desarrollo de aplicaciones para BOINC

- BOINC permite distribuir cualquier tipo de aplicación escrita en cualquier lenguaje.
  Para ello se puede hacer uso del cliente nativo directamente o encapsular la comunicación a través del *BOINC wrapper,* el cual se encargará de correr las aplicaciones como subprocesos y gestionar toda la comunicación con BOINC.

- Se puede preparar las versiones de aplicación para definir el mejor modo de correr la aplicación, esto es típicamente el configurar los entornos de ejecución como:
  - Multicore
  - OpenCL CPU
  - CUDA
  - OPENCL GPU
  - De uso no-intensivo de CPU
  - Sobre una máquina virtual
  - Con uso de MPI

- Para direccionar la comunicación, BOINC expone una serie de APIs especializadas para su gestión, estas son típicamente:
  - De inicio y datos de estado
  - De diagnóstico
  - De mensajería
  - De carga intermedia
  - De comunicación de red
  - De gestión del *wrapper*

- Permite la generación de gráficos en tiempo real para interaccionar con el usuario
- Permite la customización de los schedulers:
    - Elección de host
    - Uso de recursos (# de CPUs o GPUs, cantidad de memoria)
    - Velocidad de ejecución

- Para cada Sistema Operativo, existen una serie de reglas y librerías que asociar a la compilación de la aplicación para poder correr en BOINC **[46]**.

8.5 Creación de proyectos

- Un proyecto BOINC típicamente es uno o varios servidores con una base de datos MySQL, una estructura de directorios y un ficheros de configuración que especifica opciones, los daemons en marcha y las tareas periódicas.

- Un proyecto se define y puede encontrarse por su *master URL*, la ip de entrada.

- La estructura de directorios debe tener una estructura obligada definida como framework para alojar y solicitar los archivos necesarios de ejecución **[47]**

- La configuración del sistema reside en los siguientes puntos:
    - opciones del proyecto, típicamente un “config.xml”
    - los daemons, servicios que correrán en nuestro servidor atendiendo diferentes peticiones, por ejemplo la creación de jobs.
    - Un servicio *cron* que permita la puesta en marcha de los daemons necesarios u otros servicios requeridos.

8.6 Entrega y gestión de jobs

- El proceso convencional de entrega y ejecución de jobs consiste en un ejecutable más ficheros de entrada. El sistema remoto corre el job y devuelve los ficheros de salida cuando termina.
  Una vez el servidor recibe los resultados, este delega su validación, al daemon *validator,* y asimilación, al daemon *assimilator*.

- Los ficheros enviados/recibidos se estandarizan en una estructura xml con todos los parámetros necesarios para la entrega y gestión del job **[48]**

8.7 Experiéncia de uso

BOINC es un sistema para computación Volunteer estable. Para este trabajo se ha generado una red Volunteer local usando este ecosistema, creando todas las partes necesarias para su funcionamiento a escala local:

- Se ha descargado el servidor como máquina virtual de BOINC**[49]**.
- Se ha configurado y puesto en marcha los servicios BOINC para ejecutar una aplicación de muestra, “example_app”, la cual tiene versiones de aplicación para macOS, Win y Linux tanto para 32 como 64 bits**[50]**.
- Se configura el cliente de BOINC Manager en Windows para acceder al servidor local y este empieza a recibir y ejecutar jobs con normalidad.

Con esta configuración, ya se podría avanzar en la línea de crear el propio proyecto, no obstante, en este TFM se quiere inversigar su uso sobre arquitecturas RISC, y en concreto las ARM de los dispositivos Android, por lo que se han realizado las siguientes tareas para llevarlo a cabo:

- Compilación de las librerías nativas del cliente de BOINC para Android.
- Modificación para direccionar la App de Android hacia el proyecto en el servidor local.
- Compilación de “example_app” junto con las librerías nativas de Android para poder ser ejecutada sobre Android.**[51]**
- Añadida nueva plataforma y versión de aplicación al servidor. Esto hace que ahora reconozca las arquitecturas ARM Android y les pueda gestionar jobs.
- Se lanza la App y se empieza a recibir y ejecutar jobs con normalidad.

Otra línea de ejecución que se ha probado ha sido la configuración con dockers**[dockers]**. Esta variante se puede usar para entornos MacOS, Windows y Linux, ya que el BOINC manager creará un OS virtual donde reproducirá la condiciones de trabajo solicitadas. Además, para la creación del servidor es inmediato, ya que el docker posibilita el adjuntar el servidor en estado que se desee.

9. Conclusiones

El trabajo se formuló bajo el precepto de desconocimiento de una infraestructura sólida de computación Volunteer Grid sobre arquitecturas RISC y ARM.

Gracias a los primeros pasos del proyecto, donde se genera una tabla comparativa de las soluciones existentes, se observa de la importancia de BOINC como middleware para la mayoría de ellos, sobretodo para los proyectos académicos y para todos los proyectos con participación de clientes con arquitecturas ARM con Android.

Se ha investigado este sistema para llevar a cabo proyectos personales y conocer hasta que punto es reutilizable, viendo que es factible montar una configuración estable y lanzar un proyecto bajo una licencia de uso libre.

En este sentido se concluye que BOINC debe ser altamente considerado para la creación y gestión de redes Grid Volunteer donde los nodos pueden disponer de cualquier arquitectura.

Paralelamente, se han lanzado dos investigaciones más:

La primera busca resumir los principios sobre los que se basa la actual computación Grid, ayudando a entender el funcionamiento interno de estas redes, su arquitectura y sus requisitos.
A través de estos principios se da una visión de cómo ha de ser una arquitectura Grid sin dar a conocer detalles específicos de su implementación así como tampoco protocolos que pueden ser más propios de aplicaciones específicas.

La segunda busca conocer el tipo de nodo sobre ARM que podría correr computación Grid Volunteer, mostrando que el desgaste derivado de su uso es demasiado pequeño para tenerse en consideración, por lo que es factible llevar este paradigma a estos dispositivos siempre que se respeten las políticas de ejecución locales.

Ha quedado finalmente fuera del abasto de este proyecto la ejecución de benchmarkings comparativos entre diferentes plataformas frente a dispositivos ARM.

10. Glosario

ARM (Advanced RISC Machines):
Família de microprocesadores de arquitectura RISC bajo un mismo conjunto de instrucciones enfocados en la reducción de consumo de este.

RISC (Reduced Instruction Set Computer):
Arquitectura de microprocesador enfocada a la reducción de transistores. Esto los convierte en más baratos y eficientes energéticamente.

CISC (Complex Instruction Set Computer):
Enfocado en reducir el número de ciclos de procesador por operación. Són más caros y consumen más energía.

Commodity hardware:
Término usado para referenciar equipos de uso común.

Coarse-grained:
Define la granularidad de una tarea paralela. En este caso, el “grano grueso” se atribuye a tareas que necesitan poca comunicación entre ellas y que disponen de tramos de proceso individuales muy intensos.

HTC (High Throughput Computing / High Throughput Performance):
Computación de alto rendimiento basada en la entrega massiva de resultados.

Job:
Unidad de trabajo computacional usado en los entornos Grid. Es assignado por un scheduler.

Scheduler:
Planificador de tareas. Se encarga de decidir cuando y a quién entregar un job.

GZip:
GNU Zip. Programario libre para la compresión de datos. Usado en la transmisión de datos de BOINC.

11. Bibliografía

[1] Global smartphone sales by operating system from 2009 to 2015 (in million) **https://www.statista.com/statistics/263445/global-smartphone-sales-by-operating-system-since-2009/**

[2] ***Put your Android device to work on World Community Grid!*****. 22 de Julio de 2013** [Consultada el 16 de Febrero del 2017]. <https://www.worldcommunitygrid.org/about_us/viewNewsArticle.do?articleId=318>

[3] Ivan Rodero Castro, Francesc Guim Bernat., "Introducción a la computación distribuida de altas prestaciones", pp. 31.

[4] Adam L. Beberg et al., "Folding@home: Lessons From Eight Years of Volunteer Distributed Computing", pp. 3-4,2009.

[5] Web del proyecto Folding@Home. [Consultada el 29 de Diciembre del 2016]. <https://folding.stanford.edu/>

[6] Web del proyecto Distributed.com. [Consultada el 2 de Enero del 2017]. <http://www.distributed.net/Main_Page>

[7] Web del proyecto Samsung Power Sleep. [Consultada el 2 de Enero del 2017]. <http://www.samsung.com/at/microsite/powersleep/>

[8] ***HTC Power To Give requirements*** [Consultada el 24 de Febrero del 2017]. <http://www.htc.com/us/go/power-to-give-faqs/#FAQ6>

[9] ***World Community Grid requirements*** [Consultada el 24 de Febrero del 2017]. <https://www.worldcommunitygrid.org/help/viewTopic.do?shortName=minimumreq>

[10] ***HTCondor*** [Consultada el 10 de Marzo del 2017]. <https://research.cs.wisc.edu/htcondor/>

[11] ***Portable Batch System*** [Consultada el 10 de Marzo del 2017]. <https://www.nas.nasa.gov/hecc/support/kb/portable-batch-system-(pbs)-overview_126.html>

[12] ***Sun Grid Engine*** [Consultada el 10 de Marzo del 2017]. <http://star.mit.edu/cluster/docs/0.92rc2/guides/sge.html>

[13] Fran Berman, Geoffrey C.Fox, Anthony J. G. Hey (2003). “The anatomy of grid” pp. 178 - *Grid Computing - Making the Global Infrastructure a Reality.* England:Wiley

[14] Vladimir Silva (2006). “Enterprise computing”, pp. 32 - *Grid Computing for developers.* Massachusetts:Charles River Media, Inc.

[15] *BOINC validation* [Consultada el 20 de Abril del 2017]. <https://boinc.berkeley.edu/trac/wiki/ValidationIntro>

[16] *Open MPI* [Consultada el 15 de Abril del 2017]. <http://www.open-mpi.org>

[17] *Aspectos que influyen en la batería* [Consultada el 1 de Marzo del 2017]. <http://computerhoy.com/noticias/moviles/que-influye-autonomia-duracion-baterias-movil-34579>

[18] *Coste de consumo en la carga de las baterías móviles* [Consultada el 1 de Marzo del 2017]. <https://www.xataka.com/moviles/no-imaginas-el-dinero-que-te-cuesta-cargar-el-smartphone>

[19] *Eficiencia energética ARM* [Consultada el 1 de Marzo del 2017]. <http://www.androidauthority.com/arms-secret-recipe-for-power-efficient-processing-409850/>

[20] *eMMC* [Consultada el 11 de Marzo del 2017]. <http://www.samsung.com/semiconductor/products/flash-storage/emmc/>

[21] *What is eMMC* [Consultada el 11 de Marzo del 2017]. <https://www.datalight.com/solutions/technologies/emmc/what-is-emmc>

[22] *Understanding Flash* [Consultada el 11 de Marzo del 2017]. <https://flashdba.com/2014/06/20/understanding-flash-blocks-pages-and-program-erases/>

[23 *Standby power summary table* [Consultada el 12 de Marzo del 2017]. <http://standby.lbl.gov/summary-table.html>

[24] *Tarifa Luz España* [Consultada el 12 de Marzo del 2017]. <http://tarifaluzhora.es/>

[25] *RISC Architectures* [Consultada el 28 de Febrero del 2017]. <https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/>

[26] ARM big.Little Processing [Consultada el 28 de Febrero del 2017]. <https://www.arm.com/products/processors/technologies/biglittleprocessing.php>

[27] ARM Dynamic Scale Voltage [Consultada el 28 de Febrero del 2017].
<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0375a/Cegbfdjb.html
>

[28] ***ARM Mobile Computing*** [Consultada el 28 de Febrero del 2017].
<https://www.arm.com/markets/mobile>

[29] ***Arm devices*** [Consultada el 28 de Febrero del 2017].
<http://armdevices.net/>

[30] ***Android 4.4 Kit-Kat specs*** [Consultada el 28 de Febrero del 2017].
<https://static.googleusercontent.com/media/source.android.com/en//compatibility/4.4/android-4.4-cdd.pdf>

[31] ***Folding@home Android Client requirements*** [Consultada el 28 de Febrero del 2017].
<https://folding.stanford.edu/home/first-full-version-of-our-foldinghome-client-for-android-mobile-phones>

[32] ***World Community Grid ARM client requirements*** [Consultada el 28 de Febrero del 2017].
<https://www.worldcommunitygrid.org/help/viewTopic.do?shortName=minimumreq>

[33] ***Android 4.1 Jelly Bean requirements*** [Consultada el 28 de Febrero del 2017].
<http://static.googleusercontent.com/media/source.android.com/en//compatibility/4.1/android-4.1-cdd.pdf>

[34] ***BOINC Android client requirements*** [Consultada el 28 de Febrero del 2017].
<http://boinc.berkeley.edu/wiki/Android_FAQ>

[35] ***Android 2.3 Ginger Bread requirements*** [Consultada el 28 de Febrero del 2017].
<http://static.googleusercontent.com/media/source.android.com/en//compatibility/2.3/android-2.3-cdd.pdf>

[36] ***Distributed.net ARM client requirements*** [Consultada el 28 de Febrero del 2017].
<https://www.distributed.net/Download_clients>

[37] ***ARM ABI*** [Consultada el 28 de Febrero del 2017].
<https://wiki.debian.org/ArmEabiPort>

[38] ***ARM VFP*** [Consultada el 28 de Febrero del 2017].
<https://www.arm.com/products/processors/technologies/vector-floating-point.php>

[39] ***ARM NEON*** [Consultada el 28 de Febrero del 2017].
<https://www.arm.com/products/processors/technologies/neon.php>

[40] ***Android OS version distribution*** [Consultada el 28 de Febrero del 2017].
<https://developer.android.com/about/dashboards/index.html>

[41] ***ARMv8*** [Consultada el 28 de Febrero del 2017].
<https://www.arm.com/products/processors/armv8-architecture.php>

[42] Web del proyecto Einstein@home. [Consultada el 20 de Febrero del 2017].
<https://einsteinathome.org/es/home>

[43] Web del proyecto SETI@home. [Consultada el 20 de Febrero del 2017].
<https://setiathome.berkeley.edu/>

[44] Web del proyecto BOINC. [Consultada el 16 de Febrero del 2017].
<http://www.distributed.net/Main_Page>

[45] Wiki de Whetstone benchmark. [Consultada el 9 de Junio del 2017].
<https://en.wikipedia.org/wiki/Whetstone_(benchmark)>

[46] Building BOINC applications. [Consultada el 6 de Mayo del 2017].
<http://boinc.berkeley.edu/trac/wiki/CompileApp>

[47] Server directory structure. [Consultada el 6 de Mayo del 2017].
<http://boinc.berkeley.edu/trac/wiki/ServerDirs>

[48] Input and output templates. [Consultada el 6 de Mayo del 2017].
<http://boinc.berkeley.edu/trac/wiki/JobTemplates>

[49] VM Server. [Consultada el 6 de Mayo del 2017].
<http://boinc.berkeley.edu/trac/wiki/VmServer>

[50] BOINC/Example app. [Consultada el 15 de Mayo del 2017].
<https://github.com/BOINC/boinc/tree/master/samples/example_app>

[51] Android NDK. [Consultada el 15 de Mayo del 2017].
<https://developer.android.com/ndk/downloads/index.html>

[52] BOINC and Docker. [Consultada el 27 de Abril del 2017].
<http://boinc.berkeley.edu/trac/wiki/BoincDocker>

Otras fuentes consultadas:

Smartphone Grids [Consultada el 11 de Enero del 2017]. <https://sciencenode.org/feature/smartphone-grids-future-distributed-computing.php>

Victor Ndegwa, "Middleware for Grid Computing on Mobile Phones", 2011.

Mobile Devices: A New Source of Scientific Computing Power. 22 de Julio de 2013 por David Anderson [Consultada el 16 de Febrero del 2017]. <https://www.ibm.com/blogs/citizen-ibm/2013/07/mobile-devices-a-new-source-of-scientific-computing-power.html>

Smartphone grids - the future for distributed computing?. [Consultada el 18 de Febrero del 2017]. <https://sciencenode.org/feature/smartphone-grids-future-distributed-computing.php>

Fran Berman, Geoffrey C.Fox, Anthony J. G. Hey (2003). "The evolution of grid" - *Grid Computing - Making the Global Infraestructure a Reality.* England:Wiley

.

12. Anexos

Annexo 1: plataformas cliente < < http://boinc.berkeley.edu/trac/wiki/BoincPlatforms >

Name	Description
windows_intelx86	Microsoft Windows (98 or later) running on an Intel x86-compatible CPU
windows_x86_64	Microsoft Windows running on an AMD x86_64 or Intel EM64T CPU
i686-pc-linux-gnu	Linux running on an Intel x86-compatible CPU
x86_64-pc-linux-gnu	Linux running on an AMD x86_64 or Intel EM64T CPU
x86_64_phi-pc-linux-gnu	Linux running on a Xeon Phi
powerpc-linux-gnu	Linux running on a 32-bit PowerPC processor
ppc64-linux-gnu	Linux running on a 64-bit PowerPC processor
alpha-hp-linux-gnu	Linux running on Alpha
ia64-linux-gnu	Linux running on IA64 (Itanium)
sparc-sun-linux-gnu	Linux running on SPARC
sparc64-sun-linux-gnu	Linux running on SPARC 64-bit
arm-unknown-linux-gnueabihf	Linux running on ARM, hardware FP
arm-unknown-linux-gnueabisf	Linux running on ARM, software FP
aarch64-unknown-linux-gnu	Linux running on 64-bit ARM
powerpc-apple-darwin	Mac OS X 10.3 or later running on Motorola PowerPC
i686-apple-darwin	Mac OS 10.4+ running on an Intel CPU
x86_64-apple-darwin	Mac OS 10.5+ running on an Intel 64-bit CPU
sparc-sun-solaris2.7	Solaris 2.7 running on a SPARC-compatible CPU
sparc-sun-solaris	Solaris 2.8+ running on a SPARC-compatible CPU
sparc64-sun-solaris	Solaris 2.8+ running on a SPARC 64-bit CPU
hppa-hp-hpux	HPUX running on 32-bit HPPA
hppa64-hp-hpux	HPUX running on 64-bit HPPA
alpha-hp-tru64	Tru64 Unix running on Alpha
ia64-hp-hpux	HPUX running on IA64
powerpc-ibm-aix	AIX running on PowerPC
i686-pc-freebsd	FreeBSD on x86
x86_64-pc-freebsd	FreeBSD on Intel-compatible 64-bit
i686-pc-openbsd	OpenBSD on x86
x86_64-pc-openbsd	OpenBSD on Intel-compatible 64-bit
i686-pc-solaris	Solaris 2.8+ on an Intel x86-compatible CPU
x86_64-pc-solaris	Solaris 2.8+ on an AMD x86_64 or Intel EM64T CPU
i586-pc-haiku	Haiku on an Intel x86-compatible CPU
powerpc64-ps3-linux-gnu	Sony Playstation 3 (Cell processor) running Linux
arm-android-linux-gnu	Android running on ARM armeabi-v7a
aarch64-android-linux-gnu	Android running on ARM 64-bit arm64-v8a
x86-android-linux-gnu	Android running on Intel x86 compatible
x86_64-android-linux-gnu	Android running on Intel x64 compatible
mips-android-linux-gnu	Android running on MIPS (Big Endian)
mips64-android-linux-gnu	Android running on MIPS 64-bit (Big Endian)
mipsel-android-linux-gnu	Android running on MIPS (Little Endian)
mips64el-android-linux-gnu	Android running on MIPS 64-bit (Little Endian)