CIS1630IN02

Grid Móvil para procesar imágenes médicas

David Francisco Calle Restrepo
Alfredo Sebastián Santamaría Gómez
David Felipe Suárez Guerrero

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA DE SISTEMAS
BOGOTÁ, D.C.
2016

CIS1630IN02
Grid Móvil para procesar imágenes médicas

Autores:
David Francisco Calle Restrepo
Alfredo Sebastián Santamaría Gómez
David Felipe Suárez Guerrero

MEMORIA DEL TRABAJO DE GRADO REALIZADO PARA CUMPLIR UNO DE LOS REQUISITOS PARA OPTAR AL TITULO DE INGENIERO DE SISTEMAS

Director
Ing. Mariela Josefina Curiel Huérfano. PhD
Jurados del Trabajo de Grado
<Nombres y Apellidos Completos del Jurado >
<Nombres y Apellidos Completos del Jurado >
Página web del Trabajo de Grado
http://pegasus.javeriana.edu.co/~CIS1630IN02

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA DE SISTEMAS
BOGOTÁ, D.C.
Noviembre 2016

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERIA
CARRERA DE INGENIERIA DE SISTEMAS

Rector Magnífico

Jorge Humberto Peláez Piedrahita, S.J.

Decano Académico Facultad de Ingeniería

Ingeniero Jorge Luis Sánchez Téllez

Director de la Carrera de Ingeniería de Sistemas

Ingeniera Mariela Josefina Curiel

Director Departamento de Ingeniería de Sistemas

Ingeniero Efraín Ortiz Pabón

Artículo 23 de la Resolución No. 1 de Junio de 1946

"La Universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Sólo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la Justicia"

AGRADECIMIENTOS

Quisiéramos agradecer a nuestras familias, compañeros y profesores que estuvieron con nosotros durante nuestro desarrollo en la universidad. Especial agradecimiento a la Ingeniera Mariela Curiel por toda su ayuda y por siempre abrirnos un espacio en su agenda, también al Ingeniero Leonardo Flórez por sus útiles consejos.

CONTENIDO

FIGURAS

TABLAS

ABSTRACT

Medical image processing helps health professionals make decisions to diagnose and treat patients. Some of these algorithms require large amounts of resources, this is why they can be supported by distributed computing and an abundant number of idle mobile devices. In a previous project, Boinc was selected as the infrastructure for the Mobile Grid, however, it was required to modify the algorithms that would be executed in the devices, in order to integrate them with de system. This project addressed this problem along with the cross compilation of ITK library for the ARM architecture and the division of images to be processed in parallel.

RESUMEN

El procesamiento de imágenes médicas ayuda a los profesionales de la medicina a tomar decisiones de diagnóstico y tratamiento de pacientes. Algunos de estos algoritmos requieren gran cantidad de recursos, por esto se pueden apoyar en la computación distribuida y la abundancia de dispositivos móviles ociosos. En un trabajo anterior, se seleccionó Boinc como Grid Móvil, no obstante, se requería modificar los algoritmos a ejecutar en dispositivos móviles para integrarlos a esta infraestructura. En el presente proyecto se abordó dicho problema junto con la compilación cruzada de la librería ITK para la arquitectura ARM y la división de imágenes para su procesamiento paralelo.

I - INTRODUCCIÓN

El procesamiento de imágenes médicas es un problema computacional ampliamente estudiado que sirve de apoyo a los profesionales de la medicina cuando deben tomar decisiones con respecto al diagnóstico y tratamiento de pacientes. Algunos algoritmos para procesar imágenes requieren de una gran cantidad de recursos, con el fin de terminar en un tiempo aceptable [1], [2]. Es por ello que se apoyan en los paradigmas de computación paralela [3] y en estrategias como divide y conquistarás.

La estrategia de dividir y conquistar, consiste en segmentar el problema original en dos o más sub-problemas que requieren de un tipo de procesamiento idéntico. Cabe resaltar que, cada uno de estos sub-problemas es una instancia del problema original. La ejecución se puede realizar en distintas máquinas de forma paralela [4], [3], [5]. A este tipo de procesamiento paralelo donde se usan conjuntos diferentes de datos con el mismo algoritmo también se le conoce como el modelo de paralelización SIMD (por sus siglas en inglés Single Instruction Multiple Data) [6], [7]. Por ejemplo, en el trabajo Computing Connected Components on Parallel Computers [8], se propone solucionar, usando la estrategia de dividir y conquistar, el problema de etiquetado de regiones, que busca colocar una etiqueta a cada uno de los componentes conexos en una imagen binaria. Otro ejemplo se puede encontrar en [9]; este artículo se centra en explicar los beneficios de utilizar la estrategia de dividir y conquistar para procesar imágenes médicas, mencionando ciertos ejemplos ilustrativos de por qué la estrategia es más útil en tareas de alto nivel, analizando algunos requerimientos de dichas tareas y deduciendo qué implicaciones tiene en la arquitectura tanto de hardware como de software.

El procesamiento paralelo se inició con supercomputadores de gran tamaño y costo [10]. Pero desde 1990 se comenzaron a utilizar redes de computadores por el incremento de la capacidad, tanto de las redes, como de los computadores personales y estaciones de trabajo [5]. Aparecen los clusters de computadores débilmente acoplados que trabajan en estrecha colaboración, de modo que en algunos aspectos pueden considerarse como un solo equipo. El paso siguiente fue el paralelismo en entornos distribuidos, donde el objetivo principal es aprovechar recursos de máquinas ubicadas en distintos puntos geográficos. Esto dio origen al paradigma de computación Grid [11].

La Grid se define como una arquitectura de hardware y software que permite distribuir la carga de procesamiento entre múltiples recursos heterogéneos que pueden estar ubicados en diferentes puntos geográficos y pertenecer a diferentes organizaciones. En una Grid, los usuarios pueden acceder a distintos recursos de cómputo con poco o ningún conocimiento de dónde están ubicados o qué tecnologías están utilizando [12], creando la ilusión de un sistema de cómputo integrado. Esta técnica se entiende si se

toma como ejemplo las redes eléctricas donde, los usuarios pueden acceder a la electricidad a través de tomas de corriente sin necesidad de conocer los detalles de cómo realmente se genera.

Dada la expansión en el uso de tecnologías móviles y la evolución de sus capacidades [13], [14], [15], la idea de Grid tradicional se puede extender integrando estas tecnologías. Ello da lugar al concepto de Grid Móvil que hace posible que los usuarios accedan y ofrezcan recursos a la Grid desde sus dispositivos móviles, mejorando la capacidad de cómputo local [16], [17]. Existe una alta probabilidad de que los dispositivos móviles pudieran tener periodos de tiempo ociosos, los cuales pueden usarse para colaborar en una Grid. La incorporación de los dispositivos móviles a la Grid ofrece ciertas ventajas sobre otros recursos tales como un supercomputador o un computador de escritorio; una de estas ventajas es la ubicuidad, otra es que ocupan menos espacio. Estas ventajas pueden ser aprovechadas por diversos tipos de aplicaciones paralelas, en particular por los algoritmos de procesamiento de imágenes.

En este sentido se pudiera pensar en un centro de salud donde se utilice la capacidad ociosa de los dispositivos móviles del personal, para realizar un determinado procesamiento. Esto traería diversas ventajas al centro y al personal de la salud, por ejemplo: a) pudiera resultar más económico en término de inversiones (menos computadores físicos), b) se economizaría en espacio c) si el procesamiento está al alcance de un clic, un médico pudiera tener el diagnóstico de un paciente sin interrumpir la interacción con él.

No obstante, la tecnología de Grid Móvil está aún inmadura. Muchos desafíos provienen de las características de los dispositivos móviles: heterogeneidad, capacidad de CPU, tamaño de la pantalla, vida corta de la batería, movilidad, desconexiones intermitentes, entre otros. Adicionalmente, las redes inalámbricas (Wireless) se caracterizan por anchos de banda limitados, baja confiabilidad y altas latencias. Por otro lado, la ejecución de tareas en Grid Móviles presenta otros retos como el tipo de aplicaciones que serían más adecuadas, la seguridad y la disposición que pueden tener los usuarios a prestar sus recursos.

Aunque existen varias implementaciones de Grids móviles como IBIS [18], MoGrid [19], Akogrimo [20], MORE [21], MiPeG [22], Mobile OGSI.NET [23], estas soluciones no cuentan con una buena documentación, no ofrecen soporte porque han dejado de ser actualizadas o no están disponibles para su uso. Por las razones mencionadas anteriormente y tomando como referencia la investigación de Grid Accesibles [24], para este Trabajo de Grado se decidió utilizar y extender la tecnología Boinc [25] con el fin de desplegar una Grid Móvil en la cual se ejecuten algoritmos que procesen imágenes médicas y se basen en la estrategia de dividir y conquistar.

Sin embargo, configurar Boinc para ejecutar el clásico "Hola mundo" en un dispositivo móvil no es una tarea sencilla; requiere, además de la modificación del código fuente para introducir las llamadas necesarias al API, de ciertos conocimientos técnicos para configurar la herramienta, generar el código ejecutable compilado estáticamente, es decir, auto contenido, y la implementación de algunos algoritmos, tanto para dividir y planificar trabajos como para recolectar, unir y procesar cada resultado generado; estos algoritmos deben extenderse y comunicarse con Boinc para administrar eficientemente los trabajos en la Grid. Por otro lado, la ejecución de algoritmos de procesamiento de imágenes puede requerir de la resolución de problemas adicionales como la compilación en su versión estática de la librería ITK [26], [27] para la plataforma Android y el manejo de las diferentes partes de la imagen para su procesamiento paralelo.

El objetivo principal de este Trabajo de Grado de grado es extender Boinc para ejecutar algoritmos, previamente implementados, que procesan imágenes médicas y están basados en la estrategia dividir y conquistar.

En este documento se describen todas las tareas realizadas para lograr el objetivo planteado. El documento se estructura de la siguiente forma: en el capítulo II - DESCRIPCION GENERAL se detalla el planteamiento y formulación del trabajo realizado así como su impacto en los distintos ámbitos; en el capítulo III – MARCO TEÓRICO se describen los conceptos fundamentales utilizados y necesarios para comprender el desarrollo del trabajo; en el capítulo IV – MARCO METODOLÓGICO se explican las distintas metodologías utilizadas y cómo fueron aplicadas en cada fase del proyecto; en el capítulo V – BOINC se abordó el primer objetivo del trabajo que comprendía la familiarización con Boinc, allí se describe su arquitectura, cada uno de sus componentes y cómo se relacionan, esta sección termina con la configuración y uso de un servidor Boinc aplicando lo que se había comprendido; en el capítulo VI – WRAPPER se evalúan las posibles formas para extender Boinc y se comparan utilizando al metodología DAR, además se describe el wrapper que provee Boinc y cómo se modificó con el fin de generar su versión compilada estática para Android logrando así la primera extensión a Boinc; en el capítulo VII – ITK se detallan los problemas y soluciones encontrados para poder construir estáticamente para Android la librería ITK, requisito primordial para poder procesar imágenes médica en dispositivos móviles; en el capítulo VIII – PROCESAMIENTO PARALELO DE IMÁGENES se explica el diseño de los dos componentes principales que se implementaron para lograr la segunda extensión que se le realizó a Boinc; en el capítulo IX – PRUEBAS DE DESEMPEÑO se detallan y evalúan las distintas configuraciones de Grid Móvil que se usaron para procesar una imagen médica, comparándolas entre sí y con el tiempo de procesamiento que tardó un solo computador de escritorio en procesar la misma imagen; finalmente en el capítulo X – CONCLUSIONES se presentan los hallazgos del Trabajo de Grado al igual que los posibles trabajo futuros.

II - DESCRIPCION GENERAL

1. Oportunidad, Problemática, Antecedentes

Realizar el cómputo necesario para procesar una imagen médica implica que los médicos expertos puedan tomar decisiones acertadas en momentos críticos, que pueden significar la vida de los pacientes. Al retomar el problema de espacio, recursos y disponibilidad de computadores especializados para realizar estas tareas, en contextos como hospitales, es necesario buscar alternativas que cumplan con estas limitantes.

Se ha definido la Grid como una arquitectura de hardware y software que permite distribuir la carga de procesamiento entre múltiples recursos heterogéneos que pueden estar ubicados en diferentes puntos geográficos y pertenecer a diferentes organizaciones. Dada la expansión en el uso de tecnologías móviles y la evolución de sus capacidades [13]–[15] la idea de Grid tradicional se puede extender integrando estas tecnologías. Ello da lugar al concepto de Grid Móvil que hace posible que los usuarios accedan y ofrezcan recursos a la Grid desde sus dispositivos móviles, mejorando así la capacidad de cómputo local [16], [17].

La incorporación de los dispositivos móviles a la Grid ofrece ciertas ventajas sobre otros recursos tales como un supercomputador o un computador de escritorio; algunas de estas ventajas son la ubicuidad, un menor costo y el hecho que ocupan menos espacio. Estas ventajas pueden ser aprovechadas para el procesamiento de imágenes médicas en instituciones de salud o en zonas rurales. Además, es una solución que aprovecha la gran cantidad de dispositivos que potencialmente pueden estar disponibles para dividir un trabajo complejo en sub-tareas.

No obstante, el uso de estos dispositivos en Grids o sistemas distribuidos para procesamiento intensivo, es un área relativamente nueva de estudio que presenta varias oportunidades y retos. Las principales que se describen en Grid Computing on Mobile Devices [28] son:

- **Retos:**
    - Batería: Los dispositivos móviles tienen poca capacidad de batería y tareas de alta demanda computacional pueden agotarla rápidamente.

- Plataforma: El sistema debe ser capaz de integrar dispositivos móviles con diferentes especificaciones y sistemas operativos.
- Red: Los dispositivos móviles, usualmente, están intermitentemente conectados a diferentes redes inalámbricas.
- Legal: Lidiar con el hecho de que los usuarios no necesariamente están obligados a permitir que sus dispositivos móviles sean parte de la Grid.
- Diversidad de lenguajes: Los algoritmos existentes hacen uso de distintos lenguajes de programación y herramientas lo cual dificulta su integración. En particular para procesar imágenes médicas la mayoría de algoritmos están implementados en C++ utilizando la librería ITK [26], [27].

- **Oportunidades:**
  - CPU: La mayoría de dispositivos móviles tienen actualmente, al menos 2 GHz de velocidad de procesador y más de 1 GB de RAM. Estas especificaciones están en aumento [13].
  - Conectividad: Actualmente, existe mayor cobertura de redes con tecnologías como GPRS, 3G, 4G y Wi-Fi.
  - Almacenamiento: Aumento significativo de espacio. Registrando hasta 128 GB de espacio en disco.
  - Sensores: Uso de sensores puede ser útil para conocer el contexto en aplicaciones distribuidas.
  - Ubicuidad: Permite que los dispositivos móviles puedan colaborar sin restricciones geográficas.
  - Disponibilidad: Un dispositivo móvil generalmente está encendido y la mayor parte de este tiempo se utiliza únicamente una pequeña fracción del poder de cómputo [28], [29].

En este Trabajo de Grado se busca dar continuidad al Trabajo de Grado "Grid Accesibles" [24] donde se inició el proyecto para el procesamiento de imágenes médicas en dispositivos móviles. En el mismo se concluyó que era viable configurar una infraestructura Grid Móvil utilizando el sistema Boinc, el cual se seleccionó luego de comparar varias implementaciones de Grid. Como parte del Trabajo de Grado se plantearon tres tareas futuras: 1) desplegar el sistema utilizando la infraestructura de la universidad, 2) extender Boinc con el fin de facilitar la ejecución de aplicaciones desarrolladas para la arquitectura Android y 3) crear un sistema que permitiera la administración la Grid.

El presente trabajo continua con la segunda línea, tratando de simplificar el proceso de configurar la Grid Boinc y la ejecución de distintas tareas. A su vez, se iniciará el proceso de ejecución de algoritmos que procesan imágenes médicas usando la librería ITK (un estándar de facto que impuso las restricciones del proyecto: uso del lenguaje C++) [26], [27] y están basados en la estrategia dividir y conquistar.

1.1. Formulación del problema

¿Cómo simplificar el proceso de ejecución de algoritmos que procesan imágenes médicas basados en la estrategia dividir y conquistar sobre una Grid Móvil?

1.2. Impacto Esperado

El presente trabajo hace parte de un proyecto de investigación que tiene como objetivo el procesamiento de imágenes médicas en dispositivos móviles, por lo cual podría tener un impacto en el campo de la medicina. Se contemplaría la posibilidad de aprovechar los recursos de cómputo de múltiples dispositivos móviles, que pueden estar en una institución médica. Las aplicaciones sobre esta Grid pudieran beneficiarse de las ventajas asociadas a la computación móvil: puede alcanzar lugares y estratos sociales de difícil penetración por otro tipo de sistemas informáticos, pudiera resultar más económico en término de inversiones (menos computadores físicos), se economizaría en espacio, si el procesamiento está al alcance de un clic, un médico pudiera tener el diagnóstico de un paciente sin interrumpir la interacción con él.

Adicionalmente se hará un importante aporte, principalmente de investigación, a la comunidad científica de la Grid y Sistemas Distribuidos. Posiblemente se abrirá un extenso mundo de posibilidades para otro tipo de aplicaciones secuenciales o paralelas que hagan uso de la Grid. Esto puede enriquecer las líneas de investigación y la docencia en pre y postgrado.

2. Descripción del Proyecto

2.1. Objetivo general

Extender Boinc para ejecutar algoritmos, previamente implementados, que procesan imágenes médicas basados en la estrategia dividir y conquistar.

2.2. Objetivos específicos

- Familiarizarse con el uso y la configuración de Boinc.
- Diseñar una solución para extender Boinc permitiendo adaptar algoritmos con características específicas.
- Implementar la solución diseñada para extender Boinc.
- Ejecutar en Boinc algoritmos que utilicen librerías de procesamiento de imágenes médicas.
- Validar la solución propuesta.

III – MARCO TEÓRICO

En esta sección se explican los conceptos fundamentales para comprender el trabajo desarrollado.

1. Conceptos Fundamentales

1.1. Grid Móvil y procesamiento paralelo

- **Clúster:** Es un tipo de sistema de procesamiento paralelo compuesto por un conjunto de computadoras interconectadas a través de algún tipo de red, las cuales cooperan configurando un recurso que se ve como "único e integrado", más allá de la distribución física de sus componentes. Cada procesador puede tener diferente hardware y sistema operativo. Actualmente existen numerosas áreas de aplicación para los clúster tales como el cómputo científico, los sistemas industriales de tiempo real, el comercio electrónico (e-commerce), servicios web, sistemas militares o aplicaciones de seguridad crítica como el control de reactores nucleares [30].

- **Grid:** Es un tipo de sistema distribuido/paralelo que permite seleccionar, compartir e integrar recursos autónomos geográficamente distribuidos que pueden ser computadoras, software, bases de datos, instrumentos, dispositivos especiales y recursos humanos. Una Grid es una configuración colaborativa que se puede adaptar dinámicamente según lo requerido por el usuario, la disponibilidad y potencia de cómputo de los recursos conectados. De hecho a partir de una infraestructura de Grid "real" se pueden configurar varios sistemas "virtuales" que atienden requerimientos de diferentes usuarios [13], [31].

- **Grid Móvil:** Es un tipo de Grid en la cual dispositivos móviles son integrados a una Grid tradicional a través de canales inalámbricos para compartir recursos [17], [32].

- **Computación voluntaria**: Es un acuerdo en donde los voluntarios proveen recursos de computo a projectos, los cuales utilizan dichos

recursos para computacion distribuida y/o almacenamiento. Los voluntarios pueden ser personas del publico en general que tienen dispositivos con acceso a internet, u organizaciones como instituciones educativas o negocios. Es importante resaltar que la relación entre el projecto y los voluntarios es particulas dado que los voluntarios no son responsables ante el proyecto, es decir, que un voluntario puede intencionalmente retornar resultados incorrectos y el proyecto no puede disiplinar al voluntario; por su parte, los voluntarios deben confiar en que el proyecto provean aplicaciones que no dañarán su dispositivo y no invadirán su privacidad; adiconalmente deben confiar en que el proyecto provee inforación veridica acerca de el trabajo que hace y como se usaran los resultados intelectuales y por ultimo deben confiar en que el proyecto usa practicas de seguridad adecuadas para evitar ataques maliciosos a travez del proyecto [33], [34].

- **Computación Grid:** Es una forma de computación distribuida en la que una organización (empresa, universidad, etc.) utiliza sus ordenadores existentes (de escritorio y/o nodos del clúster) para manejar sus propias tareas computacionales de grandes cálculos y duración. Este tipo de computación difiere de la computación voluntaria de varias maneras:

    - Se confía en los recursos informáticos, es decir, se puede suponer que los nodos de la Grid no devolverán, intencionalmente, resultados erróneos o falsos. Por lo tanto, típicamente no hay necesidad de usar replicación.

    - Puede ser deseable que el cómputo sea totalmente invisible y fuera del control del usuario.

    - El despliegue de clientes es típicamente automatizado.

- **Boinc:** Es una infraestructura de software que inicialmente fue diseñada para la computación voluntaria pero también funciona para trabajar con computación Grid. Fue desarrollada originalmente en la Universidad de Berkeley. Es un medio que aprovecha una capacidad enorme de cómputo, utilizando, inicialmente, computadores personales alrededor del mundo y posteriormente el poder de

procesamiento de dispositivos móviles [35]. Para esto se encarga de distribuir trabajos entre los dispositivos conectados, verificando que sean capaces de procesarlos, para al final obtener los resultados de cada nodo. A esto se le conoce como computación de recurso público [25].

- **Middleware**: Del español Capa Media, es un componente de Software cuya responsabilidad yace en permitir la integración entre, al menos dos, dispositivos, lenguajes, sistemas o sistemas operativos que originalmente no se pudieran comunicar. Tomando un lugar entre las partes involucradas, oculta la heterogeneidad ofreciendo interfaces de comunicación a todas las partes y encargándose que exista una comunicación [36].

- **Wrapper:** O envoltorio, permite a los programas ejecutables no escritos para un arquitectura, herramienta o sistema operativo específico ser ejecutados en la plataforma objetivo. Si están disponibles los wrappers no es necesario reescribir, o modificar un programa para ser ejecutado, por ejemplo, en una Grid. El wrapper se encarga de actuar como intermediario (*Middleware***)** manejando toda la comunicación necesaria con la plataforma o herramienta destino. De esta forma hay tanta libertad en los lenguajes de programación como wrappers existan.

1.2. Compilación

- **Compilación Cruzada:** Es el proceso de generar un ejecutable que correrá en una plataforma distinta de donde se está construyendo [37], esto incluye convertir código no nativo a código nativo en la plataforma destino [38]. En este texto hace referencia al proceso realizado para generar un ejecutable para la arquitectura ARM V7, desde una arquitectura x86-64.

- **Compilación Estática:** Proceso de creación de un ejecutable donde todas sus librerías han sido, estáticamente, vinculadas durante el proceso de compilación, es decir, las definiciones a objetos o símbolos son resueltas al momento de generarlo, con lo cual éste es auto-contenido [39], [40].

1.3. Procesamiento de imágenes

- **Procesamiento de imágenes médicas:** Es el proceso mediante el cual se toma una imagen médica inicial presente en radiografías, tomografías, ultrasonidos entre otras y se aplican ciertos algoritmos con el fin de obtener datos relevantes del paciente y apoyar las decisiones diagnóstico o tratamiento [2], [41].

- **ITK:** (Insight Segmentation and Registration Toolkit) Esta es una librería de código abierto para el procesamiento y análisis de imágenes que funciona con el paradigma de tuberías y filtros. Es utilizada principalmente en aplicaciones médicas [26], [27].

- **Filtros:** En el procesamiento de imágenes, los filtros son algoritmos que reciben imágenes como entrada y realizan operaciones sobre éstas para dar como resultado una nueva imagen que resalta o atenúa ciertas características de la imagen inicial. Por ejemplo, los filtros pueden reducir ruido y suavizar o resaltar las curvas de la imagen. Las imágenes se pueden filtrar ya sea en el domino de la frecuencia o en el dominio espacial. En el domino espacial, la mayoría de los filtros operan sobre cada pixel independientemente, modificando su valor dada una matriz operadora (kernel) y los valores de los pixeles vecinos [42]. Por otro lado, en el dominio de la frecuencia no se opera independientemente sobre cada pixel sino sobre varios valores como su intensidad.

- **Filtros Gaussianos:** Estos son filtros lineales en el domino espacial que modifican la entrada mediante una convolución con la función Gaussiana, es decir, los pesos de la matriz operadora están dados por la distribución normal con varianza (σ^2). Tienen la propiedad de ser separables, es decir, cada componente se opera de forma independiente [42], [43].

2. Trabajos Relacionados

Los dispositivos móviles se han utilizado ampliamente en diversos contextos médicos. La mayoría de las aplicaciones móviles desarrolladas ayudan a gestionar y monitorear enfermedades y usualmente hacen uso de sensores de los dispositivos para recopilar información de los usuarios [44], [45]. Estos proyectos, sin embargo, no han hecho uso de los recursos que los dispositivos móviles ofrecen para resolver problemas de forma distribuida, que pudieran aprovechar una infraestructura de Grid Móvil.

El concepto de Grid Móvil no es nuevo, y por esta razón existen varias implementaciones desarrolladas en torno a este concepto, a saber:

Mobile OGSI.NET: OGSI (Open Grid Servive Infrastructure) es una plataforma para computación Grid sobre .NET desarrollada por el Grupo de computación Grid de la universidad de Virginia [46]. Define una serie de comportamientos para servicios web que son relevantes para la computación Grid. Marty Humphrey de la universidad de Berkeley, en el trabajo Mobile OGSI.NET: Grid Computing on Mobile Devices [22], [23], [47] planteó una extensión de esta plataforma para incorporar dispositivos móviles; esto lo hizo tomando diseños para computadores de alto desempeño y adaptándolos a dispositivos móviles.

MADAM (Mobility and Adaptation Enabling Middleware): Es un middleware creado por el Simula Research Laboratory de Noruega, que facilita la formación de Grids móviles y ubicuas, solucionando los problemas de heterogeneidad de los dispositivos y su naturaleza dinámica. [48].

MISCO: Es un *framework* para computación distribuida con dispositivos móviles implementado en Python. En este sistema se utiliza el modelo Map Reduce para procesar y generar conjuntos de datos muy grandes [49]. En este modelo se especifica una función *map* que procesa parejas llave/valor y genera valores intermedios y una función de *reduce* para unir estos valores (muchos problemas pueden ser expresados utilizando este modelo) [50]. Con este proyecto se espera hacer uso de las características de los dispositivos móviles como cámara, micrófono, GPS, y otra información que puede ser procesada localmente, en donde se obtuvo [49], [51].

ELASTIC: Es un proyecto realizado para el curso Parallel Computer Architecture and Programming en la universidad Carnegie Mellon en Pittsburgh. Es un *framework* que permite ejecutar tareas en paralelo con múltiples teléfonos inteligentes con sistema operativo Android. Esta plataforma se probó con una aplicación para descifrar contraseñas a partir de fuerza bruta obteniendo resultados positivos [52].

Boinc (The Berkeley Open Infrastructure for Network Computing): Es una plataforma de código abierto desarrollada por U.C Berkeley, que permite crear proyectos que operan con recursos de computación públicos. Fue creada para computación voluntaria donde, individuos que poseen máquinas con distintos sistemas operativos como Windows, Linux, Mac y actualmente Android (con la inclusión de dispositivos móviles) y que tienen acceso a internet, pueden ofrecer sus recursos para participar en proyectos de investigación. Aunque esta fue concebida inicialmente como computación voluntaria, esta plataforma, también puede ser utilizada para crear una Grid exitosamente [25], [35].

A pesar de que existen muchos proyectos que permiten utilizar los recursos de los dispositivos móviles como parte de una Grid, existen muy pocos proyectos o investigaciones que propongan utilizar estos recursos para procesar imágenes médicas en dispositivos móviles y/o dentro de una infraestructura de Grid Móvil.

Quizás el ejemplo más cercano a nuestro trabajo es el la tesis de Maestría titulada: "Desarrollo de un software para el análisis de electrocardiogramas utilizando dispositivos móviles" [53]. En esta tesis se propone el desarrollo de una aplicación en dispositivos móviles con sistema operativo Android, la cual realiza las siguientes funciones: toma una imagen de una porción de un electrocardiograma, aísla la señal de ésta para su análisis y permite la transmisión de datos relevantes a otras fases del proceso de análisis que se pueden realizar en otras máquinas. Los autores logran procesar, en un dispositivo móvil, una imagen tomada en un electrocardiograma e interpretar su contenido. El uso de un dispositivo es suficiente para el problema planteado, pues el procesamiento no es tan intensivo. En el presente Trabajo de Grado se plantea una solución para procesar imágenes médicas, por ejemplo, imágenes cuyo tamaño sobrepase la capacidad de la memoria RAM de un solo computador, por lo que se requiere un procesamiento paralelo de computadores o dispositivos móviles que pudieran estar conectados en una infraestructura Grid.

IV – MARCO METODOLÓGICO

La metodología para el desarrollo del proyecto estuvo basada en *Disciplined Agile Delivery (DAD)* [54], [55]. Esta metodología se caracteriza por ser iterativa, incremental, flexible y orientada al aprendizaje y a las personas. Se conoce como un enfoque híbrido compuesto de estrategias probadas de otros métodos ágiles. Por ejemplo, de SCRUM extiende su ciclo de vida enfocado en la construcción, para dirigir el ciclo de vida de los entregables desde el inicio del proyecto, durante el desarrollo y hasta la entrega de la solución a los usuarios finales; de *Extreme Programming* (XP) incluye consejos sobre las mejores prácticas técnicas y de *Unified Process* (UP) adopta las prácticas de documentación, modelado y estrategias de liderazgo. En la figura No. 1 se tiene una vista completa de las metodologías sobre las que se apoya DAD [56].

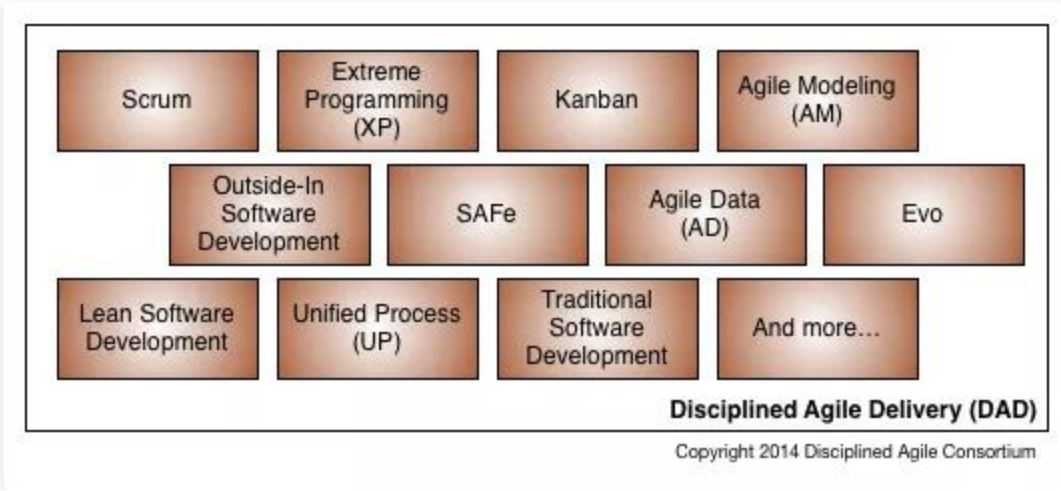

Figura 1. Conjunto de metodologías que componen DAD. Tomada de [56]

DAD tiene un ciclo de vida que toma en cuenta el riesgo-valor asociado a la entrega de un producto final, este ciclo es orientado al logro y escalable. Su ciclo de vida va desde la idea inicial del producto, a través de las entregas, operaciones y soporte. A menudo tiene varias iteraciones durante el ciclo de vida de las entregas. En la figura No. 2 se encuentra una vista de alto nivel del ciclo de vida de DAD [57].

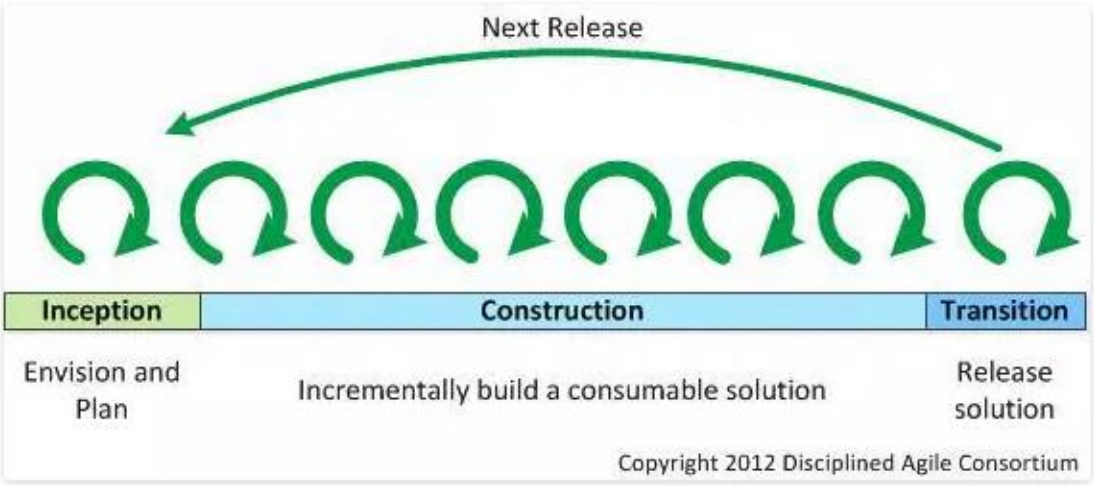

Figura 2. Ciclo de vida de DAD. Tomado de [57]

Aun así, el ciclo de vida de DAD no se restringe a lo que se visualiza en la figura No. 2. Se caracteriza por no ser rígido y se esfuerza por reflejar la realidad lo mejor que pueda, de hecho, soporta varias versiones de su ciclo de vida. Cuatro principales, a saber [58]:

- Una versión ágil-básica que se extiende el ciclo de vida de SCRUM con ideas probadas de RUP.
- Un ciclo de vida avanzado.
- Un ciclo de vida con entregas continuas.
- Una exploración del ciclo de vida "Lean Startup" [59].

DAD no prescribe un único ciclo de vida porque reconoce que un mismo ciclo de vida no se puede ajustar a todos los procesos [57]. Su gran cantidad de prácticas, técnicas y estrategias disponibles la hace fácilmente adaptable. Además, DAD ofrece consejos de cuándo y cómo aplicar dichas estrategias juntas.

Debido al carácter investigativo del Trabajo de Grado, al hecho de que existen muy pocos desarrollos relacionados con la creación y uso de una Grid Móvil y al poco conocimiento que se tenía de la infraestructura y funcionamiento de Boinc al comienzo del trabajo, se seleccionó DAD como metodología para el desarrollo del proyecto. DAD al ser una metodología orientada al aprendizaje mitiga los riesgos asociados a la capacitación de los estudiantes.

DAD se adoptó como metodología debido a que se usa comúnmente en ámbitos de investigación donde primero se debe explorar, conocer e investigar el contexto antes de tomar cualquier decisión de implementación. Ámbitos en los que se pueden tomar caminos erróneos, por lo que la metodología provee mecanismos en su fase de Concepción para evaluar opciones y tomar soluciones alternas en tiempos razonables, en el caso de que las decisiones previas sean erróneas.

Como se dijo anteriormente DAD no prescribe un único ciclo de vida. Otra de las razones por las que se seleccionó esta metodología ya que ofrece la posibilidad de adoptar el ciclo de vida que sea el más apropiado para una situación dada y luego ajustarlo adecuadamente a las necesidades. Para este Trabajo de Grado se adoptó el ciclo de vida que se extiende de SCRUM con ideas probadas de RUP. Del ciclo de vida estándar de DAD se conservó la primera fase (*Inception*) como Concepción, la segunda fase (*Construction*) se dividió en dos fases Diseño e Implementación y la tercera fase (*Transition*) además de usarla para probar otra solución, en caso de haber seleccionado una errónea, se le añadió una fase de Pruebas.

A continuación, se explican las fases de DAD utilizadas para el desarrollo del Trabajo de Grado.

1. Fase Metodológica 1 – Concepción

En la primera fase del proyecto se familiarizó con Boinc, como se planteó en el primer objetivo específico del Trabajo de Grado. Se estudiaron los componentes que Boinc usa, cómo interactúan, y cómo es el flujo básico de un trabajo que administra esta Grid. Posteriormente, se configuró un servidor Boinc con el fin de comprender la manera en la que se distribuyen trabajos en cada nodo de la Grid y cómo maneja la comunicación con cada uno de sus clientes.

Luego de tener claridad del funcionamiento de Boinc, se realizó una investigación de las alternativas que pueden extender Boinc. De esta búsqueda surgieron tres soluciones: **Construir wrapper, Modificar wrapper existente** y **Generador de código**.

Retomando el hecho que DAD ofrece mecanismos para re-evaluar soluciones, se cuenta con la posibilidad que en caso de que la primera solución que se selecciona para extender Boinc no pueda implementarse completamente para que cumpla los requerimientos establecidos o este fuera del alcance, tanto del tiempo destinado para el proyecto como de las habilidades de los estudiantes, se optará por diseñar e implementar la siguiente solución que se haya evaluado en la sección Selección de solución, generando así una nueva iteración.

2. Fase Metodológica 2 – Diseño

En esta fase se abordó el segundo objetivo específico del Trabajo de Grado que consiste en diseñar soluciones para extender Boinc. Teniendo en cuenta que se realizaron dos extensiones a Boinc, en esta fase se desarrollaron principalmente dos tareas:

- Con la primera extensión (primera tarea) se pretendía evitar la modificación de los algoritmos en C++, con el fin de facilitar su ejecución en los dispositivos que conforman la Grid Móvil. Aunque Boinc ofrece wrappers para la ejecución de algoritmos en diferentes plataformas, para el momento de comenzar este trabajo, no existía una versión compilada para dispositivos móviles y cualquier programador con intención de usar estas plataformas debía agregar llamadas al API de Boinc en su código. De modo que para resolver este problema se compararon tres soluciones que surgieron de la búsqueda de alternativas de la fase de Concepción. Para la comparación se utilizó la metodología DAR [30], [60] que consiste en identificar qué características de cada una de las soluciones deben evaluarse. Luego, dando

un peso a cada característica se realiza una matriz de comparación en la cual se puntúa cada solución. La solución que obtenga el mayor puntaje es la seleccionada para pasar a la siguiente fase de implementación.

- Con la segunda extensión de Boinc (segunda tarea) se buscó abstraer el detalle de configuración y funcionamiento de Boinc con el fin de simplificar su proceso de administración de trabajos. Se establecieron los requerimientos funcionales y no funcionales que debe cumplir esta segunda extensión. Con base en los requerimientos se diseñaron dos componentes principales: un generador de trabajos que se encarga de generar las tareas que se distribuyen en la Grid Móvil y un manejador de resultados (asimilador) de trabajos que se encarga de recolectar los resultados generados por cada tarea.

3. Fase Metodológica 3 – Implementación

Durante esta fase se desarrollaron las dos extensiones propuestas en la fase de Diseño cumpliendo así el tercer objetivo específico del Trabajo de Grado.

Para poder llevar a cabo la solución seleccionada como primera extensión de Boinc en la fase de Diseño, fue necesario conocer el funcionamiento del wrapper que provee Boinc. Luego de comprender cómo configurar los archivos que necesitaba dicho wrapper, fue necesario recurrir a la compilación cruzada con el fin de generar la versión del wrapper compilada estáticamente para Android.

Para implementar la segunda extensión se desarrollaron los componentes diseñados en la fase anterior. Ambos componentes se implementaron en Python. El generador de trabajos de desarrollo en su totalidad mientras que el asimilador de trabajos se extendió del código que provee Boinc. En el desarrollo de ambos componentes, se abstrajo el detalle de la interacción con Boinc. Evitan que el usuario deba configurar y crear los componentes y archivos que necesita Boinc, deba pensar en la comunicación con el API de Boinc y simplifican el proceso de recolección de los resultados.

Para poder ejecutar los algoritmos que procesan imágenes en la plataforma Android, se debe obtener una versión croscompilada de la librería que usan estos algoritmos. Por esta razón, durante esta fase se tuvo que resolver el problema de obtener la versión compilada estáticamente para Android de la librería ITK.

4. Fase Metodológica 4 – Pruebas

Como primera tarea de esta fase se validó el correcto funcionamiento del wrapper y del generador y asimilador de trabajos. Para esto se proceó un filtro en varios dispositivos móviles como se estableció en el cuarto objetivo específico del Trabajo de Grado. Por último, se realizaron pruebas de desempeño comparando el tiempo que tomaba procesar una imagen en varias configuraciones de Grid Móvil y variando parámetros como la cantidad y tipo de nodos de ejecución, el ambiente de la red y la cantidad de trabajos generados. Estos resultados también se compararon con el tiempo que tarda un solo computador de escritorio en procesar la imagen.

V – BOINC

1. Descripción

Boinc es una infraestructura de software que inicialmente fue diseñada para computación voluntaria pero también funciona para trabajar con computación Grid. Fue desarrollada originalmente en la Universidad de Berkeley y actualmente existen proyectos usando Boinc en diversos campos como la física, medicina nuclear, climatología, astronomía, etc. [25]. Boinc es un sistema que aprovecha una capacidad enorme de cómputo, utilizando, inicialmente, computadores personales alrededor del mundo y posteriormente el poder de procesamiento de dispositivos móviles [35]. Boinc está diseñado para ser una plataforma libre para cualquier persona natural o jurídica que quiera crear y/o colaborar en un proyecto de computación distribuida.

Se implementó para dar soporte a aplicaciones con grandes requerimientos de cómputo, almacenamiento o ambas. Adicionalmente, las aplicaciones que usan Boinc deben cumplir el siguiente requisito: ser divisibles en un gran número (miles o millones) de trabajos que pueden resolverse de forma independiente [61]. Las típicas aplicaciones que usan Boinc incluyen:

- Las simulaciones de sistemas físicos
- El análisis de cálculo intensivo de grandes conjuntos de datos
- La exploración de grandes espacios de búsqueda (incluyendo algoritmo genético).

2. Principales Características

Las características de Boinc se mencionan a continuación:

- **Autonomía de los proyectos:** Un computador puede alojar varios proyectos y aplicaciones; por seguridad y facilidad de uso cada proyecto es independiente ya que opera en su propia base de datos y sección del servidor.
- **Flexibilidad en la participación de proyectos:** Cualquier persona, natural o jurídica puede participar en múltiples proyectos; ellos son los responsables de controlar en qué proyectos participan y en cómo se dividen sus recursos entre estos proyectos. Cuando un proyecto se ha reducido o no tiene trabajos, los recursos de sus voluntarios o participantes se dividen entre otros proyectos.
- **Seguridad:** Hace uso de distintos mecanismos de seguridad para proteger a los participantes de esta Grid contra varios tipos de ataques. Por ejemplo, generando un hash de los archivos de entrada con el fin de verificar la integridad de los mismos.
- **Rendimiento del servidor y escalabilidad:** La manera en la que se programó el servidor Boinc lo hace extremadamente eficiente. Un único servidor , por ejemplo, puede enviar y manejar millones de trabajos por día. Adicionalmente, la arquitectura del servidor de Boinc es altamente escalable, por lo que es posible aumentar su capacidad o disponibilidad mediante la adición de más máquinas.
- **Disponibilidad del código fuente:** Boinc se distribuye bajo la licencia GNU Lesser General Public License [62]. Por el uso de esta licencia, a pesar de que las aplicaciones se desarrollen utilizando o extendiendo Boinc, éstas no tienen que ser de código abierto.
- **Compatibilidad con gran cantidad de datos:** Boinc es compatible con aplicaciones que producen o consumen grandes cantidades de datos, o que utilizan grandes cantidades de memoria. La distribución y recolección de datos se puede realizar a través de varios servidores. Los participantes, quienes ponen sus recursos a disposición de Boinc, pueden especificar límites en el uso de disco y ancho de banda al momento de decidir colaborar en un determinado proyecto, por lo que un trabajo se distribuye solo a las computadoras máquinas capaces de manejarlo.
- **Múltiples plataformas:** El cliente Boinc está disponible para la mayoría de plataformas más comunes como Mac OS X, Windows, Linux, Android y otros sistemas Unix.

- **Arquitectura de software abierta extensible:** Boinc proporciona interfaces documentadas de varios de sus componentes clave, por lo que es posible que terceros puedan crear software y sitios web que se extienden Boinc.
- **Comunidad de participantes:** Boinc proporciona herramientas web, tales como tableros de mensajes, perfiles de usuario y mensajes privados, que fomentan la formación de comunidades en línea.

3. Arquitectura

Boinc se basa en la arquitectura cliente-servidor, cuyo servidor o se ejecuta sobre un servidor web apache. En general el despliegue hace referencia a dos nodos principales, el cliente y el servidor. Para este trabajo se usó un VPS (por sus siglas en inglés, Virtual Private Server) con sistema operativo Ubuntu 14.04 LTS; sobre éste está instalado Apache v2.4.7 y MySQL 5.7. Dentro de Apache se corren varios procesos en segundo plano de Boinc cada uno con una funcionalidad diferente los cuales se detallan en la siguiente sección (4 Componentes).

La segunda parte del sistema hace referencia al cliente, que en este caso es un dispositivo móvil con sistema operativo Android mayor o igual a 4.1. Dentro de este teléfono está instalada la aplicación Boinc la cual se encarga de pedirle trabajos al servidor. En este Trabajo de Grado el despliegue que se quiere lograr es el que se visualiza en la figura No. 3 y tiene dos componentes principales del lado del cliente: el wrapper, que es el encargado de la comunicación con el servidor Boinc y una aplicación con la librería ITK, que es la que procesa la imagen.

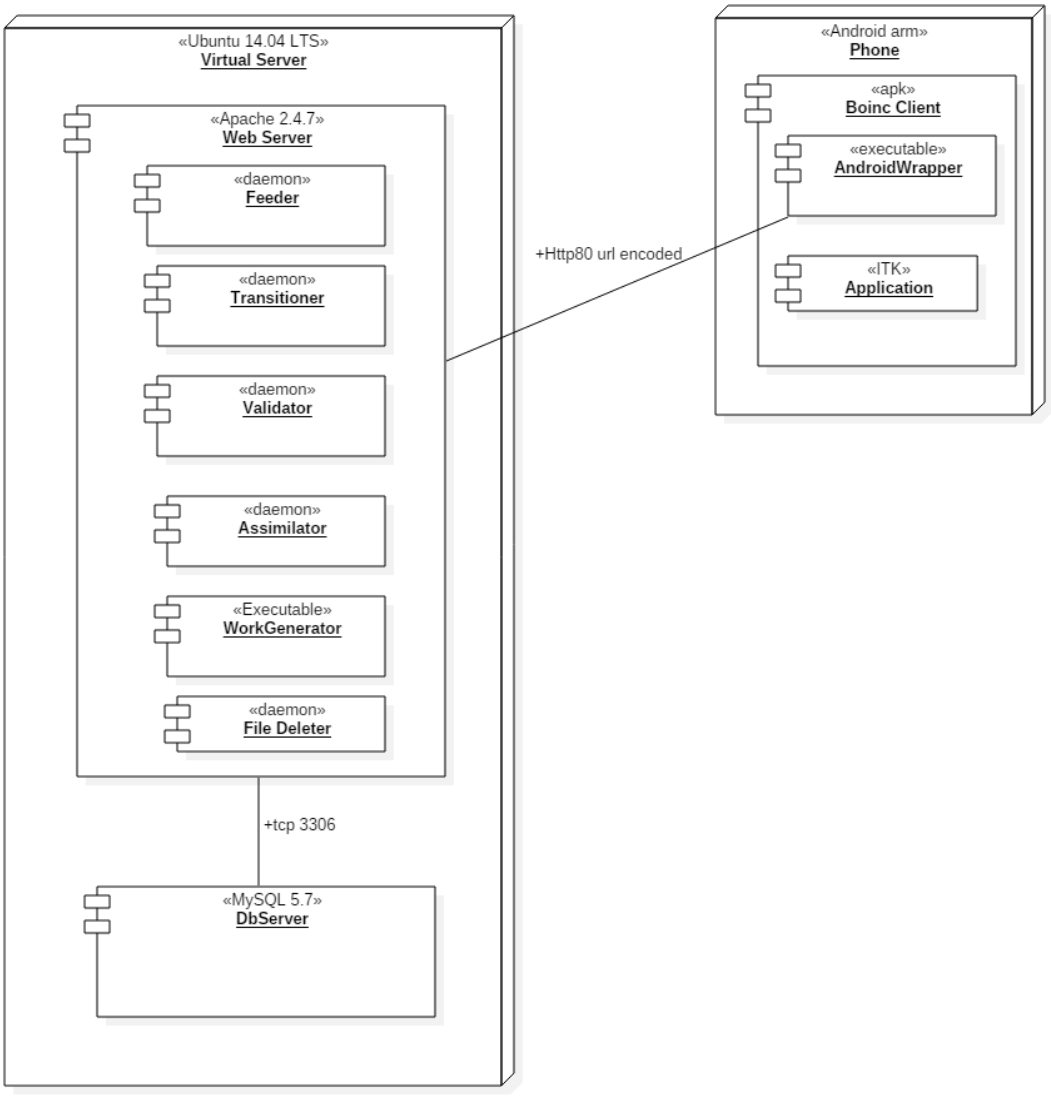

Figura 3. Arquitectura de Boinc.

4. Componentes

Boinc permite distribuir y administrar casi cualquier programa de aplicación existente, escrito en cualquier lenguaje de programación junto con sus datos, basándose en el modelo de paralelización SIMD [6], [7]. También se apoya en los siguientes componentes:

4.1. De Estructura

Estos componentes se encargan de estructurar los archivos y programas necesarios para poder ejecutar una aplicación en Boinc. Algunos de estos componentes se representan por medio de directorios como se puede observar en la figura No. 4.

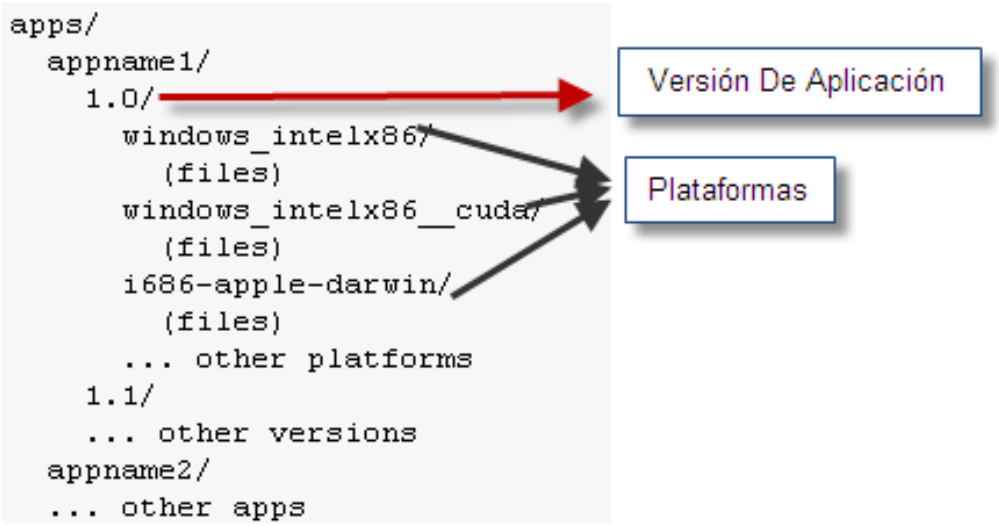

Figura 4. Componentes de estructura de Boinc. Tomado de [63]

4.1.1. Proyecto

Los proyectos son independientes; cada uno tiene sus propias aplicaciones, bases de datos, sitio web y servidores, y no es afectado por el estado de otros proyectos. Cada proyecto es identificado por una dirección URL maestra, la URL de su sitio web. Múltiples proyectos pueden coexistir en un único equipo servidor; esto puede ser útil cuando se necesite crear proyectos separados para prueba y depuración [64]. Un proyecto Boinc implica una serie de entidades relacionadas entre sí, las cuales se describen en los siguientes numerales en esta sección. Cada tipo de entidad es representada por su propia tabla en la base de datos.

En términos de implementación, un proyecto consiste en:

- Una estructura de directorios, que contiene los archivos relacionados con el proyecto.
- Una base de datos MySQL.
- Un archivo de configuración que especifica opciones, demonios, y tareas periódicas.

La forma más fácil de crear un proyecto es usando el comando **`make_project`**, que crea todas las entidades mencionadas anteriormente [64]. Este proceso se explica detalladamente en el Anexo 1: Tutorial para crear un proyecto Boinc.

4.1.2. URL maestra

Cada proyecto es identificado públicamente por una URL maestra la cual tiene principalmente dos funciones:

- Cuando se accede a ella por medio de un navegador, actúa como la página de inicio del proyecto; en ella se describe el proyecto y contiene enlaces para descargarlo en los clientes Boinc.

  Contiene elementos XML (etiquetas) de la forma que se muestra en la figura No. 5. Estas etiquetas dan las ubicaciones de los servidores que administran el proyecto. El comando **`make_project`** crea una página maestra del proyecto ubicada en *project/html/usuario/index.php*. Este archivo incluye el archivo ***'schedulers.txt'***, que contiene la lista de **<scheduler>** que posee el proyecto actualmente [65]. Estas etiquetas pueden ser incorporadas dentro de comentarios HTML. El cliente Boinc lee y analiza la página maestra para localizar el servidor que contiene el proyecto.

```
<scheduler> http://host.domain.edu/cgi/scheduler
</scheduler>

<scheduler>
http://host2.domain.edu/cgi/scheduler</scheduler>
```

Figura 5. Archivo *'schedulers.txt'* de un proyecto Boinc.

4.1.3. Aplicación

Una aplicación incluye varios programas con su respectiva versión para distintas plataformas, un conjunto de unidades de trabajo (**workunits**, en términos de Boinc) y resultados. Un proyecto puede incluir múltiples aplicaciones [66].

4.1.4. Plataforma

Es una combinación de arquitectura del CPU y un sistema operativo. Boinc define un conjunto de plataformas estándar para que máquinas con diferentes sistemas operativos colaboren en un proyecto, formado una Grid Heterogénea. Se debe crear un directorio independiente con el nombre de dicho sistema operativo (plataforma), por ejemplo, **windows_intelx86**, permite usar dispositivos que cuenten con Windows de 32 bits. Así, al momento en el que un cliente Boinc con plataforma *x* solicita trabajos de un proyecto, se le envía el ejecutable de la aplicación compilado para la plataforma *x* tan sólo dirigiéndose a la carpeta bajo el nombre de la plataforma *x*. Boinc soporta 41 plataformas en donde se encuentran diferentes configuraciones de Windows, Linux, Mac Os, Solaris, OpenBSD y Android entre otras [67].

4.1.5. Versiones de la aplicación

El ejecutable de una aplicación puede ir a través de una secuencia de versiones. Una versión particular, compilada para una plataforma en particular, en Boinc se le conoce como versión de la aplicación. Una versión de la aplicación se compone de uno o más archivos.

Cada vez que se genera una nueva versión del ejecutable para una plataforma en particular, Boinc crea una carpeta dentro de la respectiva carpeta de la plataforma, esto con el fin de tener distintas versiones del ejecutable para cada plataforma. Es decir, se puede tener una plataforma con 3 versiones de un ejecutable y al mismo tiempo otra plataforma con 5 versiones del mismo ejecutable. Cuando un cliente Boinc solicite trabajos, se le envía la última versión del ejecutable de su respectiva plataforma.

4.1.6. Workunit

Una unidad de trabajo (**workunit**, en términos de Boinc) describe el cálculo a realizar. Cuenta con varios atributos, éstos se especifican al momento de creación: [68]

a) **Nombre:** Cadena de texto, que identifica de manera única la unidad de trabajo (workunit) en el proyecto. Con el fin de no repetir nombres de workunits el programador puede agregar el PID del proceso de creación, un número de secuencia o una marca de tiempo.

b) **Aplicación:** El nombre de la aplicación que el workunit ejecutará. Un workunit está asociado con una aplicación, no con una versión particular de la aplicación.
c) **Archivos de entrada:** Una lista de los archivos de entrada y sus nombres. Estos archivos se descargan del servidor.
d) **Batch (Opcional):** Es un entero que puede ser utilizado para operar (cancelar, cambiar la prioridad, etc.) en los grupos de workunits.

4.1.7. Resultado

Describe la instancia de un workunit, ya sea sin empezar, en curso o terminado. Cada resultado se asocia con una unidad de trabajo. En algunos casos, puede haber varios "réplicas", de una unidad de trabajo determinada. Cada resultado cuenta con los siguientes atributos:

a. Archivos de salida: Es la lista de los nombres de los archivos de salida (*physical names*), y los nombres por los que la aplicación se refiere a ellos (*logical names*).

b. Estado en el servidor (`server_state`): El cual representa el estado en el servidor y puede tomar los valores de:

- Inactivo: No está listo para enviar el resultado al servidor
- Sin enviar: Listo para enviar a un cliente, pero aun así no se ha enviado.
- En curso: Enviado, pero no se ha procesado.
- La tarea fue procesada con éxito por el cliente
- Fuera de tiempo límite
- Ejecutado con error
- No necesario: La unidad de trabajo finalizó antes de que se envió este resultado.

c. Host: Cliente Boinc que ejecutó el cálculo.

d. Estado de salida: Valor retornado al terminar el cálculo (0 si la ejecución culminó con éxito).

e. Tiempo de CPU: El tiempo de CPU que se utilizó.

f. Datos de archivo de salida: Los tamaños y las sumas de comprobación de los archivos de salida para verificar la integridad del archivo.

g. Tiempo de entrega: El momento, e términos de horas, minutos y segundos, en que se recibió el resultado.

4.2. De Interacción

Estos componentes son los encargados de administrar en la Grid los trabajos generados, actualizando la base de datos del proyecto cada vez que un trabajo cambia de estado. También se encargan de la comunicación con cada cliente Boinc enviando los trabajos generados y recibiendo los resultados. Normalmente son programas que se ejecutan en segundo plano (demonios).

4.2.1. Generador de trabajo (*WorkGenerator*)

Este es un programa encargado de crear las unidades de trabajo (workunits). Se encarga de asociar el trabajo con la aplicación (ejecutable) que los clientes procesarán, así como sus parámetros de entrada. El programador puede suministrar su propio generador de trabajos o usar el que provee Boinc por defecto. Sin embargo, es importante recalcar que el que provee Boinc es muy sencillo: crea infinitos trabajos con la misma entrada de datos. Si se quiere un esquema diferente de distribución de datos entre los trabajos es importante desarrollar un generador de trabajos propio, el cual debe programarse utilizando el API que provee Boinc. Esta tarea es complicada para cualquier programador que no conozca Boinc ya que no solo necesita aprender a usar el API, sino que debe aprender a compilar aplicaciones Boinc con C++.

4.2.2. Alimentador (*Feeder*)

Este es un programa que se encarga de revisar la base de datos del proyecto en busca de trabajos, cada vez que un cliente los solicita. Para ello, el *feeder* verifica las especificaciones técnicas del cliente, rastrea en la base de datos un trabajo que el cliente sea capaz de procesar y se lo envía junto con sus parámetros de entrada. Este programa es suministrado por Boinc y no es posible usar uno personalizado [69].

4.2.3. Wrapper

El wrapper que provee Boinc ejecuta las aplicaciones (que se quieren correr en la Grid) como subprocesos y maneja todas las llamadas necesarias al API de Boinc para mantener la comunicación con el cliente, por ejemplo, para informar el tiempo de CPU, el estado de un trabajo, cargar sus resultados, etc. Este componente se detalla en la sección Modificación del Wrapper para Android

4.2.4. Manejador de Transiciones (Transitioner)

Este es un programa que maneja las transiciones de estado tanto de las unidades de trabajo como los resultados, actualizando la tabla

correspondiente en la base de datos del proyecto. Este programa es suministrado por Boinc y no es posible usar uno personalizado. Los estados que maneja son [69]:

- Necesita validar
- Validado
- Asimilado
- Borrar archivos
- Error

4.2.5. Validador (*Validator*)

Este programa decide si los resultados de trabajos terminados son "válidos". Para esto el *validator* compara los resultados obtenidos de distintos clientes en busca del resultado que más se repita [70].
La validación de un resultado consiste en dos 2 partes:

- **Comprobación de sintaxis:** Verificar que los archivos de salida estén presentes en el servidor Boinc y que tengan el formato correcto.

- **Verificación de replicación:** Si el trabajo del cual se generó el resultado se ha marcado como réplica, éstas se comparan. Si Boinc encuentra que la mayoría de réplicas son "equivalentes", los resultados de esas réplicas se consideran como válidos y el resto inválidos. Para este tipo de validación Boinc provee un Validador con nombre SampleBitwiseValidator, que se encarga de comparar los archivos resultantes de cada réplica y revisar si son idénticos a nivel de Bits.

El programador debe especificar un validador para cada aplicación en su proyecto e incluirlo en la sección <*daemons*> del archivo de configuración del proyecto [71]. Es importante recalcar que Boinc también provee la opción de no crear réplicas para cada trabajo, en ese caso también provee un validador, cuyo nombre es *SampleTrivialValidator*, este no hace ningún tipo de validación a los resultados, solo se encarga de marcar el resultado de la tarea como válido.

4.2.6. Asimilador (*Asimilator*)

Es un programa que se encarga de manejar y procesar los resultados marcados como válidos. El procesamiento depende de lo que desee el

usuario. Podría, por ejemplo, copiar los archivos de salida que se encuentran en el directorio de carga por defecto de Boinc a una ubicación permanente antes de que el *File Deleter* los elimine; también podría analizar los archivos de salida e insertar los resultados en una base de datos [72].

4.2.7. Eliminador de Archivos (*File Deleter*)

Este programa actúa cuando todos los resultados se han marcado como **Borrar archivos**. Se encarga de eliminar todos los archivos que fueron necesarios para procesar y administrar un trabajo, por ejemplo, archivos de salida, de entrada, de configuración, plantillas (*templates*), etc.

Los componentes anteriormente descritos actúan de tal forma que el resultado que un componente genera al procesar un trabajo o resultado sirve de entrada del siguiente componente. Esta interacción es ilustrada en la figura No. 6

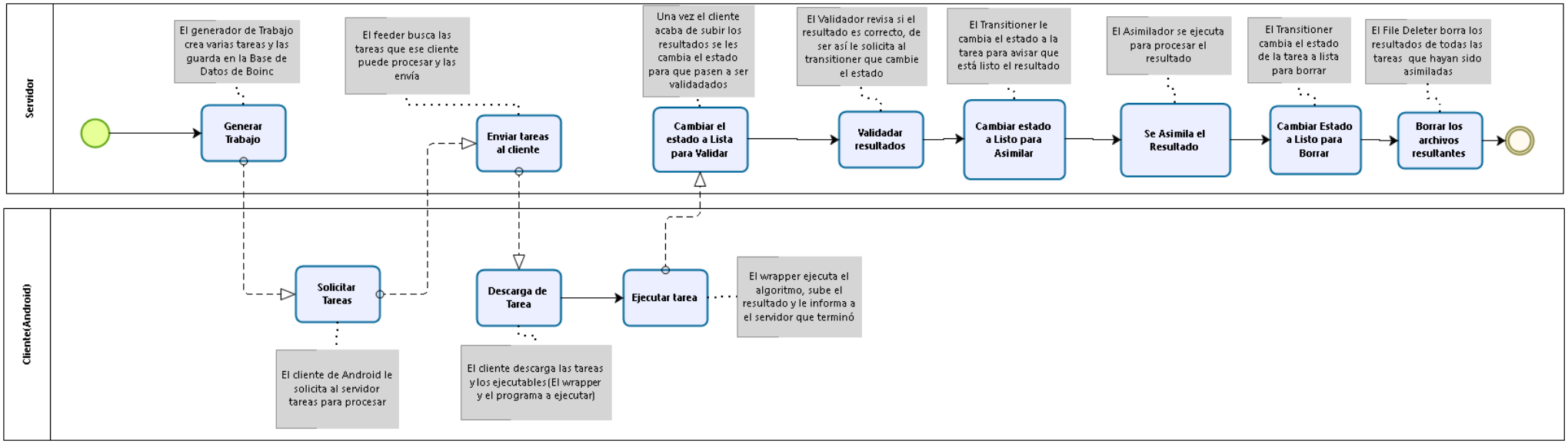

Figura 6. Interacción y flujo de los componentes de Boinc.

5. Comunicación servidor-cliente

Con el fin de tener mayor conocimiento de cómo funciona Boinc, es decir, cómo envía trabajos, recibe resultados, identifica la plataforma y características técnicas (batería, capacidad de almacenamiento y disco, capacidad de cómputo, etc.) de cada cliente, mientras mantiene una comunicación efectiva con ellos, es útil comprender el entorno en que se ejecutan aplicaciones en Boinc.

En términos generales, el flujo de Boinc y el ciclo de vida de sus trabajos se menciona a continuación [66], [73]:

- Cuando un cliente Boinc se queda sin trabajos, emite una petición al servidor que contiene uno de los proyectos al que está suscrito. El mensaje de petición especifica la plataforma del cliente.
- Un generador de trabajos (***work generator*),** crea el trabajo y sus archivos de entrada asociados.
- El *feeder* explora la base de datos del proyecto en busca de trabajos que pueden ser ejecutados por el cliente, teniendo en cuenta sus especificaciones de software (plataforma) y hardware (batería, capacidad de disco, etc.).
- El cliente descarga cada instancia del trabajo, por intermedio del *feeder* y archivos (ejecutables y de entrada) asociados a la unidad de trabajo.
- El cliente ejecuta el trabajo y sube, al servidor, los archivos de salida generados como respuesta.
- El cliente informa al servidor Boinc que la tarea se ha completado.
- El validador comprueba los archivos de salida, comparando las réplicas. Esto se hace para evitar que usuarios maliciosos entreguen resultados erróneos.
- Cuando se termina un trabajo, el asimilador se encarga procesar estos resultados.
- Una vez que todos los trabajos se han completado y se han procesado, el eliminador de archivos (*file deleter*) elimina los archivos de entrada y salida.

Este ciclo se representa en la figura No. 7 y se repite indefinidamente hasta que no existan más trabajos por procesar.

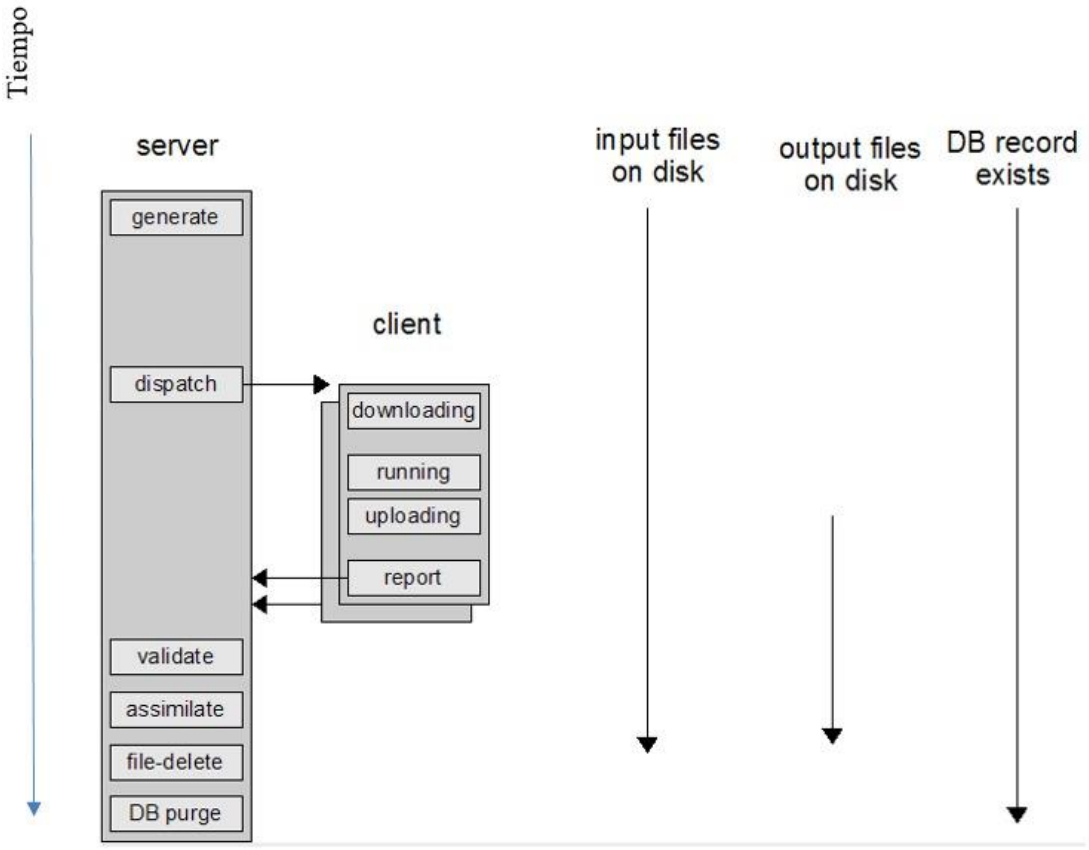

Figura 7. Ciclo de vida de un trabajo. Tomado de [68].

Para que este ciclo se realice exitosamente, Boinc mide los siguientes aspectos técnicos de cada cliente [74]:

- **Rendimiento de la CPU:** precisión de operaciones de coma flotante por segundo, operaciones enteras por segundo, y el ancho de banda de memoria. Estas medidas se toman cuando el cliente realiza un trabajo por primera vez, luego una vez cada mes.
- **Número de CPU's:** Se indican en las preferencias del usuario.
- Proveedor y el modelo de CPU.
- **Espacio de disco:** Espacio libre y espacio total en el disco. Estos números se utilizan para asegurar que Boinc no utilizará más espacio que el establecido por las preferencias del usuario.
- **Memoria:** Memoria RAM total, caché del CPU y el espacio de intercambio (*swap*). Estos números pueden ser utilizados por el servidor para decidir si asignar o no un trabajo al cliente.

- Zona horaria.
- Número de RPC´s (por sus siglas en inglés, Remote Process Call) y hora del último RPC
- Sistema operativo junto con su versión.

La comunicación entre los componentes de Boinc está implementada utilizando memoria compartida . Existe una estructura de memoria compartida para cada aplicación que se ejecute [75].

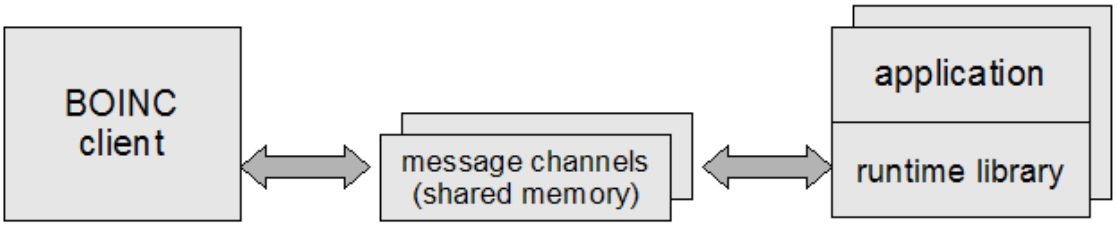

Figura 8. Comunicación cliente-servidor Boinc. Tomada de [75]

Luego de conocer cada componente de Boinc, el siguiente paso es crear y configurar un servidor Boinc, junto con cada una de sus entidades mencionadas anteriormente.

6. Configuración del Servidor

Se aprovecharon las capacidades de la computación en la nube para crear un VPS, por sus siglas en inglés Virtual Private Server utilizando los servicios que proveen Digital Ocean [76] y Amazon AWS, allí se configuró Boinc. Estos VPS se crearon con el sistema operativo Linux, distribución Ubuntu 14.04 al cual se le agregó Apache 2.4.7 y MySQL 5.7, ya con estos dos componentes funcionando se instalaron las dependencias necesarias para el funcionamiento de la Grid y se clonó el repositorio con el código fuente de Boinc. Se creó el usuario **boincadm** y se agregó al grupo **www-data**, para la ejecución y administración de la Grid con permisos para operar sobre el servidor web apache. Fue necesario otorgarle los permisos de escritura, lectura y ejecución al usuario **boincadm** sobre las carpetas que contendrán el proyecto con el que se probará el servidor Boinc. Se compiló el código fuente de Boinc, finalizando así la instalación de la Grid en el servidor. Los detalles específicos del proceso de instalación y configuración se encuentran en el Anexo 2: Configurar Servidor Boinc.

7. Uso del servidor

Ya con el servidor Boinc configurado, se decidió hacer el despliegue, primero en computadores de escritorio y luego en dispositivos móviles. De esa forma los estudiantes, autores de este trabajo, se familiarizaron con el uso de la herramienta, pasando de lo más sencillo a lo más complejo.

7.1. Ejecución de un "Hola Mundo"

Inicialmente se usó el Boinc con la clásica aplicación "¡Hola Mundo!", esta se probó con un cliente Linux instalado y configurado localmente en una máquina distinta a donde se encontraba el servidor Boinc. Posterior a este paso se siguió con el procedimiento de creación de un proyecto. Luego se generó el ejecutable para la plataforma del cliente, Linux en este caso, y por último se creó una aplicación con su respectiva versión y plataforma donde se ubicó el ejecutable generado.
Como resultado se pudo ejecutar una aplicación sencilla en distintos clientes Linux sin necesidad de modificar el código fuente de la aplicación, para agregar las llamadas necesarias a las librerías de Boinc, gracias al wrapper para Linux que provee Boinc.

7.2. Filtro Gaussiano

El siguiente paso y con el fin de empezar a incursionar en la ejecución de programas que procesen imágenes, se decidió ejecutar una implementación en ITK del filtro Gaussiano llamado *Smoothing Recursive Gaussian Image Filter*, el cual se encarga de aplicar una transparencia a la imagen que recibe como entrada, generando así una imagen un poco más desenfocada y borrosa que la original.
El proceso que se siguió fue similar al descrito en 7.1. No obstante, fue necesario compilar estáticamente la librería ITK para Linux y enlazarla con el ejecutable generado para la versión de aplicación en Boinc. Se obtuvo el mismo resultado que con la primera aplicación, la ejecución fue exitosa y gracias al wrapper que provee Boinc para Linux no fue necesario modificar los algoritmos.

Con lo anterior se resolvió la primera parte del problema: ejecutar algoritmos que procesen imágenes en la Grid Boinc, sin introducir aun dispositivos móviles. Se concluyó, entonces, que para correr cualquier tipo de algoritmo en Boinc lo único distinto al proceso estándar y donde se concentra la dificultad es en la generación del ejecutable sin carga dinámica de librerías y para la plataforma destino.

VI – WRAPPER PARA DISPOSITIVOS MÓVILES

Como se mencionó en la sección Componentes, la función del wrapper es evitar que el programador tenga que modificar el código fuente incorporando las llamadas al API de Boinc necesarias para que su programa se ejecute en la Grid. Boinc provee versiones compiladas del wrapper para varias plataformas, entre las cuales no está Android. Uno de los primeros objetivos planteados en el trabajo, se relaciona con extender/modificar Boinc para que los programadores que realicen aplicaciones para dispositivos móviles no tengan que modificar sus programas para incluir llamadas al API de Boinc o, de ser necesario, esto se haga de forma automática. En este capítulo se describen las alternativas estudiadas y seleccionadas para resolver este problema.

1. Aplicación Metodología DAR

En esta sección se listan y explican las posibles soluciones que se contemplaron durante la fase de concepción del presente trabajo, con el fin de simplificar y/o extender el uso de Boinc [35] para ejecutar algoritmos en dispositivos móviles. Posteriormente se comparan las alternativas utilizando la metodología DAR [60].

- **Construir un wrapper:** Se propone desarrollar un programa que emule el funcionamiento de un wrapper, actuando como intermediario entre el ejecutable que se quiere correr en Boinc y Boinc. Este programa se encargará de la comunicación con Boinc, evitando así la modificación del código fuente.
- **Modificar un wrapper existente:** Se realizó una búsqueda dentro del repositorio de Boinc, en la documentación disponible y en foros de la comunidad, con el fin de identificar las implementaciones existentes de distintos wrappers para Android. En esta búsqueda se encontró que no existen como tal wrappers específicos para Android. Boinc provee un wrapper "genérico" que actualmente funciona para 6 plataformas, dentro de las cuales no se encuentra la plataforma principal de Android (armv7) [77]. Con esta solución, lo que se quiere es explorar la posibilidad de adaptar el wrapper existente para que funcione en plataformas Android.
- **Generador de código:** En esta solución se plantea desarrollar un programa que modifique el código fuente del algoritmo que se ejecutará en la Grid Móvil agregando las invocaciones al API de Boinc.

Luego de identificar las posibles soluciones, se realizaron sesiones de lluvia de ideas entre los directamente interesados en el proyecto de investigación: la ingeniera Mariela Curiel, al igual que los estudiantes David Calle, Alfredo Santamaría y David Suárez. Durante las sesiones se discutieron los posibles usos futuros y otros aspectos que pudieran ser de utilidad en las fases siguientes de este Trabajo de Grado, tales como: la facilidad de administración y uso, el alcance del proyecto, la complejidad de diseñar e implementación dicha solución, como también qué tan genérica y simple de usar podría ser. Estas sesiones produjeron la siguiente lista de características que fueron evaluadas para cada solución.

1. **Facilidad de uso:** Esta característica tiene como objetivo medir el esfuerzo necesario para que un usuario final pueda ejecutar un algoritmo en Boinc usando dispositivos móviles.
2. **Curva de aprendizaje:** Es el esfuerzo necesario, por parte de los estudiantes que desarrollan el Trabajo de Grado, para poder implementar la solución.
3. **Modificación del algoritmo:** Necesidad de que el programador, explícitamente tenga que agregar líneas al código fuente del algoritmo con el fin de invocar métodos del API de Boinc. Es importante considerar esta característica porque al modificar el código se pueden introducir errores.
4. **Duración de la implementación:** Tiempo necesario para implementar la solución que cumpla con los requerimientos establecidos.

Después de seleccionadas las características a evaluar se decidió darle a cada una el mismo peso cuantitativo ya que cada atributo afecta directamente los requerimientos establecidos que debe cumplir la solución (ver Anexo 3: Requerimientos del sistema). El puntaje que se otorgó a cada solución en cada característica evaluada se visualiza en la tabla No. 1.

Características	Construir wrapper	Modificar wrapper existente	Generador de código
Facilidad de uso	3	3	4
Curva de aprendizaje	1	2	2
Modificación del algoritmo	4	4	1
Duración de la implementación	1	2	1
TOTAL	**9**	**11**	**8**

Tabla 1. Matriz comparativa DAR.

A continuación, se presentan los argumentos que explican la puntuación asignada a cada solución para cada uno de las características:

1. **Facilidad de uso:** Para esta característica el Generador de Código obtuvo la mayor puntuación ya que su uso es tan sencillo como ejecutar un programa que recibe como entrada el código que se quiere correr en la Grid Móvil y genera este mismo código, pero agregando automáticamente las invocaciones al API de Boinc. Por otra parte, las soluciones asociadas al uso de un wrapper (creación o modificación) requieren que haya un proceso de aprendizaje por parte del usuario final ya que debe aprender a usarlo y configurarlo. Por ejemplo, para usar el wrapper que provee Boinc, es necesario configurar manualmente un archivo XML con la configuración del wrapper.
2. **Curva de aprendizaje:** En esta característica la puntuación más baja es para Construir Wrapper ya que además de informarse del funcionamiento normal de un wrapper, se debe diseñar e implementar. Lo anterior implica aprender y aplicar buenas prácticas de programación y posiblemente el uso de patrones y nuevos paradigmas de programación para lograr implementar un wrapper genérico y reutilizable. Por otro lado, en las otras dos soluciones o se cuenta con una base de conocimiento y un producto del cual partir (en el caso de Modificar Wrapper Existente) o se debe desarrollar un programa que no requiere conocimientos avanzados en programación o paradigmas (en el caso del Generador de Código) ya que básicamente consiste en implementar un programa que modifique un archivo de texto plano.
3. **Modificación del algoritmo:** Para esta característica claramente las soluciones que involucran un wrapper son las ganadoras, ya que estos se encargan de la comunicación con Boinc sin modificar los algoritmos de procesamiento de imagens. Por el contario, el Generador de Código obtiene la puntuación más baja ya que éste se encargaría de explícitamente agregar al código existente las llamadas al API de Boinc.
4. **Duración de la implementación:** En esta característica el Generador de Código y Construir Wrapper obtuvieron la más baja puntuación ya que a diferencia de Modificar Wrapper no se cuenta con una base de la cual partir, sino que se debe indagar e investigar fases que podrían tomar más tiempo que entender el wrapper actual y adaptarlo.

Se evidencia que la solución que obtuvo el puntaje más alto es **Modificar Wrapper Existente.** Es una de las soluciones más deseables porque se evita la modificación directa del código fuente de la aplicación que correrá en Boinc y se aprovecha código probado mitigando así la posibilidad de error. Como se plantea utilizar un wrapper existente, se estima que la implementación se centre en la corrección y adaptación de este wrapper siendo así menos extensa que la

creación de una solución nueva (wrapper o generador de código). Por otro lado, la curva de aprendizaje podría ser un poco lenta ya que se debe leer y entender código de terceros que posiblemente no esté diseñado para plataformas Android, comprender su lógica de programación, investigar el funcionamiento y composición de librerías y comandos desconocidos para los estudiantes, pero necesarios para utilizar y compilar el wrapper.

2. Modificación del Wrapper para Android

Para que el wrapper funcione correctamente se debe crear y configurar un archivo con nombre lógico ***"job.xml"*** ya que es el primer archivo que el wrapper lee para conocer aspectos como: qué aplicación ejecutar, cómo ejecutar la aplicación que se le indica (comandos adicionales, secuencia de las tareas, parámetros de entrada, etc.) y cómo ubicar los recursos que la aplicación requiere. En el archivo ***"job.xml"*** se pueden especificar múltiples tareas, esto es útil para dos propósitos:

- Para manejar trabajos que implican múltiples etapas. Por ejemplo, pre-procesamiento y post-procesamiento.
- Para dividir una tarea larga en partes más pequeñas.

A continuación, se especifica el contenido de este archivo el cual consiste en una serie de etiquetas XML:

- **application:** Nombre lógico de la aplicación (ejecutable) que correrá el wrapper.
- **stdin_filename, stdout_filename, stderr_filename:** Nombre lógico de los archivos donde las entradas y salidas estándar **in, out** y **error** escribirán, si el código del ejecutable lo requiriera.
- **command_line:** Comandos que se agregarán a la línea de comandos con la que se invocará el ejecutable.
- **weight:** El aporte de cada tarea a la fracción de ejecución global, es proporcional al uso de recursos que requiere. Por ejemplo, si un trabajo tiene tareas A y B, y A utiliza 100 veces más el CPU que B, se debe establecer A.weight = 100 y B.weight = 1.
- **checkpoint_filename:** Nombre del archivo utilizado para escribir puntos de control manejados por la aplicación, si se requiriera. Cuando este archivo se modifica, el wrapper asume que un punto de control se ha alcanzado y notifica al cliente.

- **fraction_done_filename:** Nombre del archivo donde periódicamente se escribe la porción de la tarea que se ha procesado (es un número entre 0 a 1). Este es usado por el wrapper para reportar qué tanto de la tarea ha procesado un cliente.
- **exec_dir:** Directorio donde se debe iniciar la aplicación, puede ser relativo o utilizar la variable de entorno $PROJECT_DIR.
- **multi_process:** Si la aplicación crea múltiples procesos. **Nota: cada proceso padre debe esperar a sus hijos para terminar.**
- **setenv:** Variable de entorno que necesitan las aplicaciones. Es posible tener más de una etiqueta **<setenv>**. Se debe de usar la siguiente manera: NOMBRE DE LA VARIABLE = VALOR DE LA VARIABLE, por ejemplo:
  - LD_LIBRARY_PATH = $LD_LIBRARY_PATH
- **daemon:** Tareas que deben correr como "demonios", es decir, en segundo plano y de forma asíncrona, mientras que las otras tareas se ejecutan secuencialmente. El wrapper terminará este demonio cuando la última tarea haya terminado.
- **time_limit:** Si se especifica valor para esta etiqueta el wrapper terminará la tarea luego de transcurrido el tiempo límite establecido.
- **priority:** Prioridad de las tareas de la siguiente manera:
  - 1: Más baja (Win: IDLE; Unix: 19)
  - 2: Baja (Win: BELOW_NORMAL; Unix: 10)
  - 3: Normal (Win NORMAL; Unix: 0)
  - 4: Alta (Win: ABOVE_NORMAL; Unix: -10)
  - 5: Más alta (Win: ALTA; Unix: -16)

El wrapper puede, opcionalmente, descomprimir archivos de entrada y salida.

- **unzip_input:** Antes de ejecutar tarea alguna, el wrapper se descomprimirá los archivos de entrada especificados en esta etiqueta.
- **zip_output:** Después de que todas las tareas se han completado, el wrapper comprimirá archivos de salida especificados en esta etiqueta.

Para utilizar el wrapper fue necesario crear y configurar una serie de archivos adicionales, los cuales Boinc específica detalladamente en [77] como parte de su documentación. Adicionalmente, para familiarizarse con la manera en cómo opera este wrapper se realizaron varios ejemplos que Boinc implementó y se pueden encontrar en su repositorio de código fuente

(https://github.com/BOINC/boinc) bajo el directorio ***boinc/samples/*** los cuales Boinc explica en [78] también como parte de su documentación.

2.1. Compilación cruzada del Wrapper existente

El wrapper que provee Boinc se debe compilar para la plataforma objetivo. Sin embargo, la comunidad Boinc sólo tiene versiones compiladas para las siguientes plataformas:

- **Windows_intelx86**
- **Windows_x86_64**
- **i686-pc-linux-gnu**
- **x86_64-pc-linux-gnu**
- **i686-apple-darwin**
- **x86_64-apple-darwin**

Con esto se encontró un gran reto, que fue realizar la compilación cruzada del wrapper para ejecutarlo en plataformas Android ARM.

El primer paso para iniciar la compilación cruzada fue consultar el tutorial que ofrece Boinc [79].

Las principales características que se encontraron de este tutorial son:

- El tutorial está dirigido a aplicaciones escritas en C o C++.
- Está diseñado para un ambiente Linux.

No obstante, en dicho tutorial no hay ninguna documentación sobre cómo se debe compilar el wrapper para Android. Se podía encontrar información sobre cómo compilar aplicaciones hechas para Android, sin embargo, este no era el caso dado que el wrapper tenía su propio proceso de construcción el cual era necesario adaptar. El principal problema encontrado fue que en el tutorial hacían referencia a variables que no existían en el *makefile* del wrapper.

Se decidió buscar respuesta a los problemas generados por las inconsistencias del tutorial en la comunidad Boinc, foros o wikis, pero debido a que el tema tratado en el presente trabajo es relativamente nuevo no se obtuvo mayor ayuda.

Para superar este obstáculo se optó por leer el código fuente de Boinc y buscar dentro de los *commits* alguna referencia al proceso de construcción del wrapper. Se logró encontrar un archivo que no se mencionaba en ninguna parte de la documentación, su nombre era **build_wrapper_arm.sh**. Este archivo se encarga de la construcción del wrapper, se componía de una serie de comandos

que normalmente se ejecutan directamente en una terminal. Luego de entender el código fuente se pudo descubrir cómo funcionaba el proceso de construcción y la compilación cruzada del wrapper que provee Boinc.

Sin embargo, al intentar ejecutar el archivo se encontraron varios problemas. El primero era que para poder usar este archivo era necesario previamente configurar el *toolchain* de Android utilizando el NDK proveído por Google. Este contiene todo lo necesario para hacer desarrollo de aplicaciones Android utilizando C/C++ y era utilizado por el archivo mencionado anteriormente para construir el wrapper. Se descargó la versión más reciente del NDK y se generó el *toolchain* a partir del mismo. A pesar de que la generación del *toolchain* fue exitosa al intentar construir el wrapper salían varios errores. Se decidió probar con diferentes versiones del NDK hasta que se logró superar estos errores utilizando la versión 10e.

El siguiente problema encontrado es que el proceso exigía versiones antiguas de las librerías Curl y Openssl y que además estuvieran compiladas para Android. No fue posible encontrar versiones pre-compiladas por lo que se optó por buscar el código fuente de la versión de estas librerías e intentar hacer la compilación cruzada de las mismas.

Dentro de este nuevo proceso, el primer paso fue dirigirse al repositorio de Curl (https://github.com/curl/curl) y de Openssl (https://github.com/openssl/openssl), para descargar el código fuente de ambas librerías. Posteriormente, con ayuda de los archivos utilitarios **build_curl_arm.sh** y **build_openssl_arm.sh** de Boinc se intentó construir estas librerías. Sin embargo, a pesar de tener el código fuente de Curl en la versión exacta que exigía Boinc, había un error al construir la librería **libboincapi**: el proceso solicitaba una versión determinada de Curl y de no encontrarse paraba la construcción del wrapper. Se decidió manipular dicha validación para que siguiera con el proceso sin importar la versión de la librería. Después de este cambio se logró obtener una versión compilada para Android del wrapper genérico de Boinc, el paso a paso de este proceso se encuentra en Anexo 4: Compilar Wrapper Para Android.

3. Validación

El wrapper solo se encarga de ejecutar la aplicación que se le indique en el archivo "job.xml", la cual podría ser desde un simple "¡Hola Mundo!" hasta aplicaciones que contengan algoritmos de procesamiento de imágenes. Esto quiere decir que cualquier aplicación se debería ejecutar correctamente sin importar su complejidad, ya que el wrapper no debe verse afectado por este tipo de cambios. Por lo anterior, se decidió probar la funcionalidad del wrapper con

una aplicación sencilla donde se copiaba el contenido de un archivo que se recibía como parámetro a uno nuevo generado por la misma aplicación. La validación consistió en revisar que la salida fuera correcta.

El programa se ejecutó primero sin hacer uso del wrapper, para descartar la posibilidad de que cualquier error que se llegará a generar al momento de incorporar el wrapper, estuviera asociado al mal funcionamiento de la aplicación en el sistema operativo Android. Las subsecciones 3.1 a 3.3 explican las distintas etapas del proceso.

Es importante resaltar que en esta fase del trabajo no era posible probar con algoritmos para el procesamiento de imágenes médicas debido a que no se tenía la versión compilada de ITK para Android, problema que se resolvió y se detalla en la sección Compilación cruzada y estática.

3.1. Ejecución Manual

Como primera prueba, la aplicación mencionada anteriormente se ejecutó manualmente en el dispositivo móvil. Para ejecutarla se debía hacer primero la compilación cruzada del programa. En este proceso, era necesario utilizar el *toolchain* que ya se había configurado durante la construcción del wrapper, y escribir un archivo llamado **Android.mk**, una versión simplificada de un *makefile* para Android. Aunque se siguieron los pasos de la documentación, se generaba un error en la compilación porque no se encontraba la librería **stdio**. La solución a este problema fue modificar el **Android.mk** para que utilizara la variable *APP_STL:=stlport_static* la cual agrega la librería estándar de C al proceso de construcción.

Finalmente se pudo construir la aplicación para Android y con ayuda del SDK de Android y la herramienta de línea de comandos ADB (Android Debug Bridge) que permite acceder directamente a los archivos del sistema de un celular (conectado a un computador de escritorio vía USB) y desde allí ejecutar aplicaciones como normalmente se hace en ambientes Linux: `./nombreEjecutable`, se logró ejecutar la aplicación en el dispositivo móvil.

3.2. Ejecutable PIE

Tras haber ejecutado manualmente la aplicación de prueba sin errores, y verificando la escritura correcta del archivo de salida, se prosiguió a hacer uso del wrapper.

Al intentar ejecutarlo en dispositivos móviles Android con versión mayor o igual a 5.0, se obtenía un error en tiempo de ejecución que informaba que

el programa que no era un ejecutable PIE (por sus siglas en inglés, Position Independent Executable). Se decidió entonces probar en dispositivos móviles Android con versión menor a 5.0 en los cuales ya no se presentaba este error. Al intentar solucionarlo, se encontró que Android desde su versión 5.0 exige [80] que todos los programas que se quieran ejecutar en su plataforma debe contener la característica de seguridad PIE la cual es compatible desde la versión 4.1 del sistema operativo [81], [82].

Un código ejecutable PIE puede ser reubicado, es decir, cada vez que se ejecute el programa, este puede ser cargado en diferentes direcciones de memoria haciendo que sea mucho más difícil para un atacante aprovechar errores en un programa [83], [84].

Teniendo en cuenta la velocidad con la que avanza la tecnología y dadas las cifras de la página Statista, tomadas en septiembre del 2016 que indican que el 97% de usuarios de Android tiene una versión mayor o igual a 4.1 y cerca del 54% mayor o igual a 5.0 [85], era evidente la necesidad de tener soporte para esta característica PIE. Por esto, se modificó una vez más el proceso de construcción del wrapper, esta vez afectando directamente su código fuente para añadir las banderas "*–fPIE*" y "*–fPIE –pie*", las cuales permitieron construirlo como un ejecutable con la característica PIE y así poder utilizar dispositivos móviles Android con versión mayor o igual a 5.0.

3.3. Ejecución con Wrapper

Una vez generado el wrapper como un ejecutable PIE, se probó una vez más el wrapper en dispositivos móviles Android con versión mayor a 5.0. y nuevamente se generó un error en tiempo de ejecución. La causa de éste se debía a que la variable de entorno **PWD** (que obtiene la ruta del directorio actual) en Android no existe o no es visible para el programador. Para solucionarlo se modificó el código fuente del wrapper cambiando el llamado de **PWD** por la función ***getcwd*** qué hace parte de la librería **unistd** de C.

Con estas nuevas modificaciones fue posible utilizar el wrapper compilado para Android y ejecutar correctamente la aplicación descrita al inicio de esta sección la cual generó correctamente el archivo de salida.Con lo descrito en esta sección se solucionó la siguiente parte del problema: poder ejecutar algoritmos en dispositivos móviles haciendo uso de un wrapper.

VII – EJECUCIÓN DE ALGORITMOS PARA EL PROCESAMIENTO DE IMÁGENES MÉDICAS

Teniendo la certeza, de que ahora es posible utilizar la infraestructura proveída por Boinc para crear una aplicación que se pueda desplegar en dispositivos móviles con sistema operativo Android sin necesidad de modificar el código fuente para comunicarse con Boinc, el siguiente paso es explorar la ejecución de programas para el procesamiento de imágenes médicas, que usen la librería ITK. Para la ejecución de estos algoritmos se requirió previamente la compilación de la librería para la plataforma destino. Una vez compilada la librería se seleccionó un filtro que cumpliera con las características requeridas para el proyecto.

1. Descripción de la librería ITK

ITK es una librería para realizar registro[1] y segmentación[2] de imágenes. Esta librería en conjunto con VTK, ITK, KWWidgets, y IGSTK son las librerías de código abierto más utilizadas en el campo de procesamiento y análisis de imagines médicas. Han sido extensivamente probadas y su robustez, flexibilidad y extensibilidad está garantizada; son el estándar en esta área [86].

La librería ITK está desarrollada en C++ y garantiza soporte multi-plataforma (cross-platform) utilizando el entorno de construcción CMake [87]. Provee una alta gama de técnicas de segmentación, registro y filtrado de imágenes, pero no proporciona una interfaz gráfica o métodos para visualización de datos; sin embargo, existe un proceso establecido para su integración con la visualización proveída por VTK.

2. Compilación cruzada de ITK

Para utilizar algoritmos con la librería ITK es necesario construirla para la plataforma destino, en este caso Android, con la ayuda de la herramienta CMake. ITK consiste de múltiples subsistemas y a partir de la versión 4.0.0, lanzada en Diciembre de 2011, fue oficialmente modularizada. Actualmente consiste de más de 100 módulos internos y otros remotos, los cuales están agrupados en [27]:

- Core
- ThirdParty
- Filtering
- IO

[1] En procesamiento de imágenes registro se refiere a la tarea que busca encontrar correspondencias entre los datos.
[2] En procesamiento de imágenes médicas registro se refiere al proceso para identificar y clasificar datos encontrados en una representación muestreada digitalmente [26].

- Bridge
- Registration
- Segmentation
- Video
- Compatibility
- Remote
- External
- Numerics

La construcción de algunos de estos módulos requiere de la generación de ejecutables intermedios que deben ser ejecutados para completar la construcción del módulo exitosamente. Dado que estos ejecutables se generan para la plataforma destino, en este caso Android ARM, no se pueden ejecutar dentro de la plataforma en la cual se hace la compilación cruzada, Linux x86_64. Por esta razón, actualmente no es posible realizar la compilación cruzada de todo el kit de herramientas.

Para la realización de este Trabajo de Grado se construyeron los módulos núcleo (Core) y de filtrado (Filtering) y adicionalmente el módulo de filtro gaussiano que no pertenece a los módulos por defecto. Para la compilación cruzada también fue necesario desactivar las banderas que realizan pruebas, pues deben ejecutarse en la plataforma destino.

Por último, la aplicación no debía cargar librerías dinámicamente en el dispositivo móvil ya que no se puede asegurar que estos tengan las instalaciones necesarias para poder ejecutar los algoritmos. Como consecuencia, es necesario deshabilitar la carga de librerías en tiempo de ejecución, con el fin de generar un código ejecutable estático. Este proceso se detalla en el Anexo 5: Compilación Cruzada de ITK.

3. Selección del algoritmo

Uno de los objetivos de este trabajo es ejecutar algoritmos que utilizan la estrategia dividir y conquistar y tratan con ello de aprovechar el procesamiento paralelo. La idea es tener un paralelismo de grado mediano a grueso, con mucha independencia de las tareas paralelas trabajando sobre el conjunto de datos. Teniendo lo anterior en cuenta, no parecían adecuados aquellos filtros que requirieren de alguna forma información de toda la imagen para hacer los cálculos de una región en especial. Por ejemplo, filtros que realizan inicialmente un muestreo de toda la imagen para modificar cada uno de sus pixeles dependiendo de estos valores, pues los resultados de este filtro serían diferentes si se realizan en subregiones independientes de la imagen ya que localmente este muestreo sería diferente al global y modificarían los pixeles de forma diferente.

Se escogió un filtro con la propiedad de procesar independientemente diferentes regiones de una imagen para luego unir los resultados. El filtro a utilizar debe tomar en cuenta la información de sus vecinos haciendo posible procesar regiones independientes, los vecinos seleccionados se deciden dando un radio y tomando los que se intersecten con este. Dado que se ejecuta el mismo algoritmo sobre distintas partes de la imagen inicial, el modelo de ejecución paralela es *Single Instruction Multiple Data* (SIMD).

El filtro utilizado lo provee ITK: S*moothingRecursiveGaussianImageFilter*, genera una imagen difuminada realizando una convolución sobre la imagen, esta utiliza un kernel que tiene valores dados por la distribución Gaussiana o normal [88]. El resultado de este algoritmo se puede observar en la figura No. 9 sobre una resonancia magnética de cerebro, a la izquierda la imagen original y a la derecha la imagen difuminada después de realizar el procesamiento del filtro.

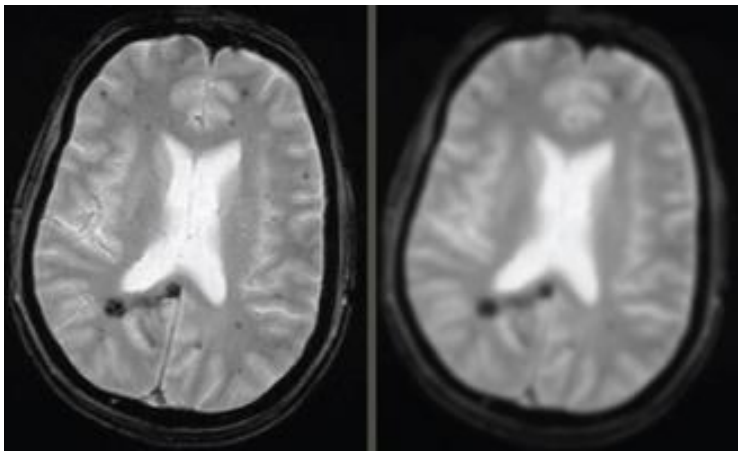

Figura 9 resultado de filtro *SmoothingRecursiveGaussianImageFilter* sobre resonancia magnética de cerebro

VIII – PROCESAMIENTO PARALELO DE IMÁGENES

Habiendo logrado compilar y ejecutar el wrapper y una aplicación de ITK para Android, el siguiente paso fue utilizar Boinc para dividir el procesamiento entre múltiples dispositivos. Para hacer esto, se encuentran dos retos: a) automatizar el proceso de creación de trabajos y b) recopilar los trabajos para generar el resultado final, ya que Boinc solo se encarga de almacenar los resultados como archivos de salida en un directorio temporal.

Para solucionar el primer problema fue necesario crear un generador de trabajos que le permitiera a un programador implementar su propio algoritmo de división de imagen y

poder integrarlo a la infraestructura Boinc, sin mayor conocimiento de su API, y sin tener que incluir la librería de Boinc en el proceso de construcción de su código. Actualmente Boinc no provee ningún componente, a parte de su librería, para poder generar trabajos.

La solución al segundo problema fue extender el manejador de resultados (asimilador) que actualmente provee Boinc. El objetivo de esta extensión fue simplificar el proceso de recolección de resultados ya que el programador no tiene que conocer los detalles del funcionamiento de Boinc.

El objetivo de esta sección es plasmar los diseños realizados, explicar cómo estos se integran con Boinc y con los dispositivos móviles con sistema operativo Android. Adicionalmente, se explica cómo un programador (usuario final), que requiera integrar sus algoritmos de división y unión de trabajos debe utilizar los componentes desarrollados en este Trabajo de Grado.

1. Diseño del Generador de Trabajos

El primer componente desarrollado es el generador de trabajos, para este desarrollo no se utilizó ningún componente desarrollado por Boinc debido a que el generador de trabajos que provee actualmente es de propósito demostrativo y no tiene funcionalidad práctica, este componente nuevo implementa el patrón adaptador, permitiendo integrar un algoritmo de división previamente implementado (el que provea el programador) con Boinc y crear trabajos sin que el programador conozca los detalles específicos del funcionamiento. En la figura No. 10 se visualiza el diagrama de clases, en el cual se puede notar la presencia de la clase *WorkDivisor*. Esta clase representa a la fachada de un algoritmo de división previamente implementado. La fachada se comunica con el adaptador y crea un trabajo dada una lista de rutas de archivos de entrada. El adaptador se encarga de comunicarse con el API del servidor Boinc y retornar el identificador del trabajo creado. Esto le servirá al programador en el futuro para poder identificar los resultados de sus trabajos.

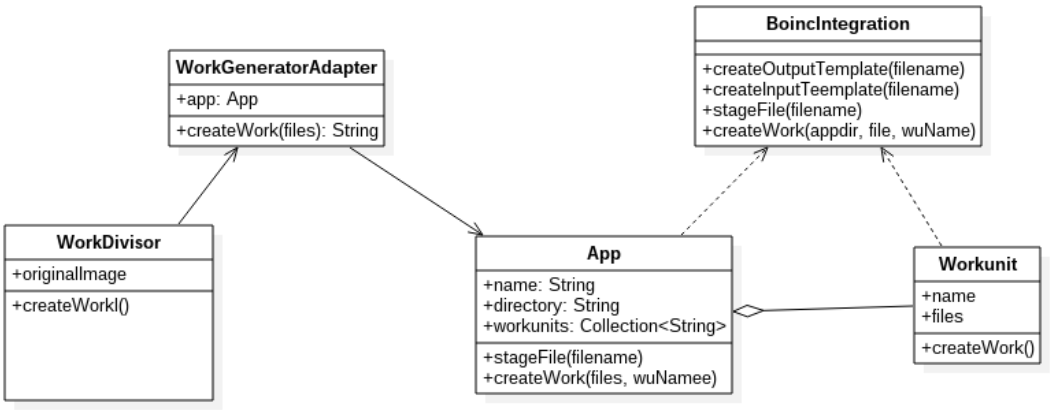

Figura 10. Diagrama de clases del Generador de Trabajos.

El flujo de trabajo principal del generador es el siguiente: el código creado por el programador, que divide la imagen utiliza el adaptador comunicándole que debe crear uno o varios trabajos a Boinc con determinados archivos de entrada, el número de archivos de entrada de cada uno de los trabajos lo debe definir el programador ya que depende la aplicación y de cómo se divide la imagen. El adaptador se encarga de comunicarse directamente con el API de Boinc e inicializa cada trabajo con un identificador único. Este flujo se ilustra en la figura No. 11.

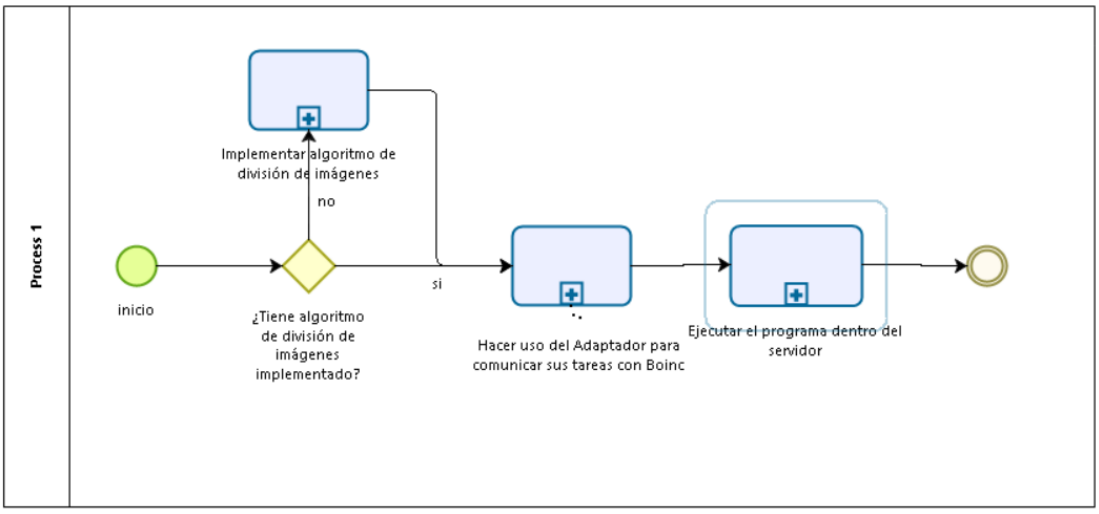

Figura 11. Diagrama de procesos Generador de Trabajos.

2. Diseño del Asimilador

El segundo componente desarrollado fue el asimilador de trabajos. El objetivo de este componente es hacer operaciones con los resultados de las unidades de trabajo terminadas, ya que una vez termina un trabajo y es validado, Boinc se encarga de borrar todos los archivos subidos por el cliente. Esto hace que sea necesario usar los resultados de las operaciones inmediatamente o guardarlos de tal forma que no haya pérdida de información. Actualmente Boinc ofrece un asimilador escrito en Python que no realiza ninguna tarea, debe ser extendido para agregarle funcionalidad. Sin embargo, se necesita saber muchos detalles del servidor para poder utilizarlo; el objetivo de este componente fue simplificar el proceso para que el programador pueda obtener los resultados fácilmente.

Debido que existen múltiples algoritmos para unir imágenes, dependiendo de la aplicación específica, se decidió crear un asimilador que fuese lo suficientemente extensible para que dados uno o muchos algoritmos de unión, previamente implementados, éstos pudieran ser adaptados al proyecto de la manera más fácil posible. Para cumplir este requerimiento se hizo una implementación del patrón observador, donde los observadores son los algoritmos de unión implementados por el programador y el observado es una clase que se encarga de detectar cuando una unidad de trabajo está lista para asimilar y notificar los cambios a los observadores para que puedan ser procesados. Es deber del programador implementar la interfaz del observador y registrarse ante el observado. En la figura No. 12 se visualiza el diagrama de clases del asimilador de trabajos.

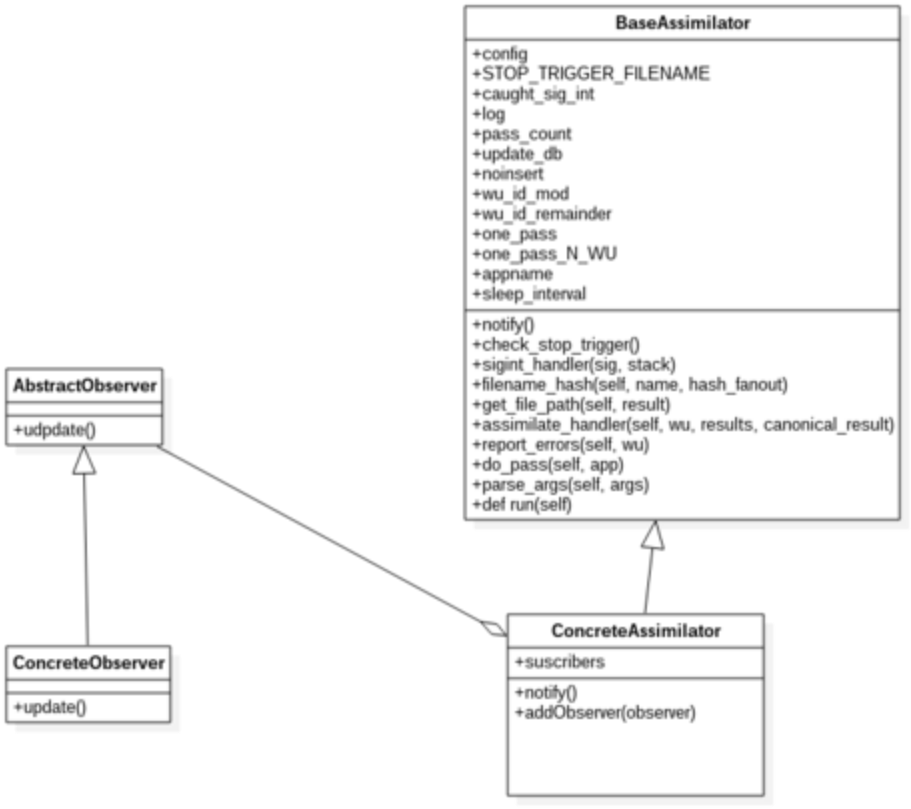

Figura 12. Diagrama de Clases Asimilador de Trabajos.

El flujo del asimilador es el siguiente: El código desarrollado por el programador se registra ante la clase *ConcreteObserver* en busca de conocer las tareas terminadas y sus resultados. La clase *ConcreteObserver* utiliza el API de Boinc para buscar las tareas terminadas; si encuentra nuevos resultados en estado Listo para Asimilar, notifica a todos los observadores y le cambia el estado a la tarea a Asimilada. Este se ilustra en la figura No. 13.

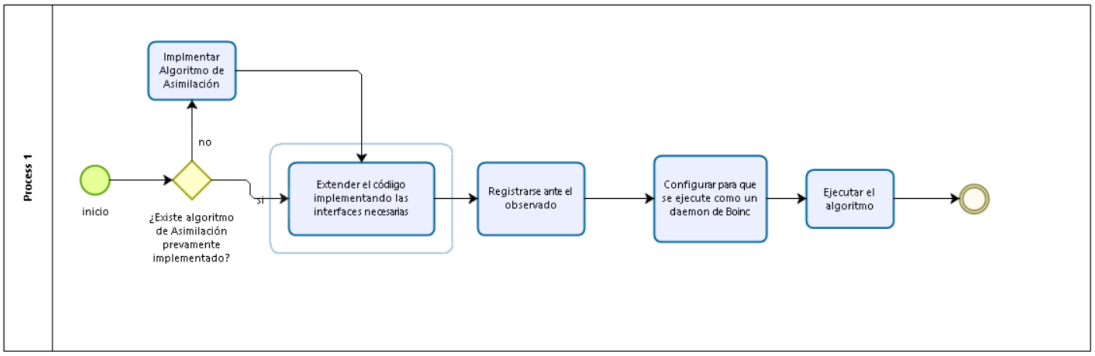

Figura 13. Diagrama de procesos Asimilador de Trabajos.

Los diseños mencionados anteriormente se detallan en el Anexo 6: Documento de Diseño de Software SDD.

3. Pruebas del Generador de Trabajos y Asimilador

Para probar el generador y asimilador de trabajos diseñados en la etapa anterior, se implementaron 2 algoritmos, uno para dividir la imagen y otro para unir los resultados. Según el diseño, estos algoritmos deben ser implementados por el programador pues dependen de la aplicación específica, la cual no se conoce a priori. El diseño realizado facilita la integración de estos algoritmos con la infraestructura de Boinc.

En este caso la división de la imagen se hizo como una cuadricula de M filas por N columnas donde, cada trabajo independientemente realizaría un proceso de difuminación en una región rectangular de la imagen. Dado que se utilizó el filtro ***SmoothingRecursiveGaussianImageFilter*** (descrito en la sección Selección de algoritmos), que utiliza el valor de los pixeles vecinos, fue necesario tomar regiones con tamaños un poco más grandes que se solapaban, con el fin de que los bordes de la región que se procesaba de la imagen no afectaran el resultado global.

Utilizando esta solución, los diferentes dispositivos realizan cálculos redundantes sobre regiones que se solapan, pero al momento de realizar la unión de los resultados, se eliminan estas regiones obteniendo un resultado exactamente igual al que se obtiene realizando la misma operación sobre la imagen completa sin divisiones, estas comparaciones se realizaron utilizando las herramientas ***imageDiff*** y ***PerceptualDiff***.

Este proceso se puede observar en la figura No. 14 donde la región punteada representa el tamaño real de la imagen que se procesa, pero en la unión de resultados únicamente se utiliza la región con línea continua, asegurando así que el procesamiento de los pixeles en los bordes utilicen los valores de sus vecinos y no cambie el resultado final al unir cada resultado.

Estos algoritmos específicos de división y unión fueron escritos en C++ utilizando la librería ITK. Es importante resaltar que estos algoritmos pueden ser escritos en cualquier lenguaje y utilizando diferentes librerías, manteniendo la misma facilidad de integración a la infraestructura que provee Boinc.

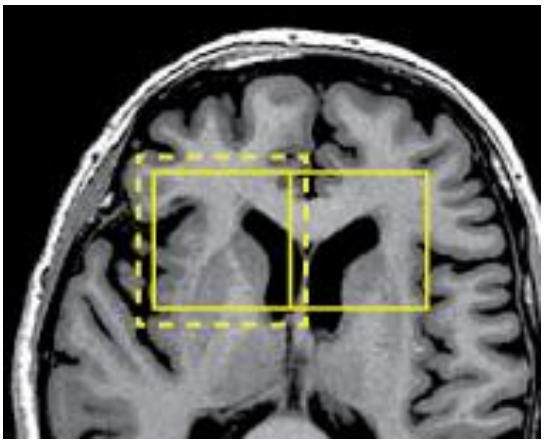

Figura 14 Solapamiento de regiones de la imagen procesada

IX – PRUEBAS DE DESEMPEÑO

En esta sección se decide hacer una evaluación preliminar de desempeño del algoritmo propuesto, ejecutándose en dispositivos móviles con la intermediación de Boinc. Se presenta un diseño experimental 2^k que permite observar y cuantificar cómo afecta un conjunto de factores el desempeño del algoritmo. Entre estos factores se encuentra el tipo de dispositivo (móvil o computador de escritorio), la ubicación del servidor Boinc y el grado de paralelización, entre otros.

1. Prueba Base

Inicialmente se realizó una prueba base con un computador con sistema operativo Linux x86_64 y procesador Intel Core i7-920XM 2.0 GHz con 8 Núcleos y 4 Gb de RAM. Para esta prueba no se utilizó la infraestructura de Boinc.

El objetivo de esta prueba fue obtener un tiempo referencial de ejecución del algoritmo *SmoothingRecursiveGaussianImageFilter* en un solo computador de

escritorio sobre una imagen en formato tiff de 301 Mb de tamaño. Esta misma imagen y algoritmo fueron los factores que no variaron en todas las pruebas realizadas y descritas en la próxima sección. El tiempo de ejecución de esta prueba fue de 49 minutos.

2. Experimento

2.1. Diseño Factorial 2^k

El diseño experimental se utiliza para determinar el efecto o impacto que tienen k factores en una prueba de desempeño, cada uno de los cuales tiene 2 alternativas o niveles [89]. Es un diseño experimental exploratorio y la idea es determinar qué factores tienen más influencia sobre la variable respuesta. Una vez determinados los factores es posible realizar más experimentos con un mayor número de niveles por factor. Este segundo grupo de experimentos está fuera del alcance de este trabajo.

En este caso se seleccionaron 4 factores relevantes para el proyecto y la variable respuesta fue el tiempo total de ejecución del algoritmo. Cada factor tiene dos niveles cada uno etiquetados como -1 o 1. Los factores seleccionados se explican a continuación:

- **Dispositivos de procesamiento (móviles o fijos):** Este factor se refiere a las características técnicas que tienen los dispositivos que se encargaron de procesar los trabajos generados. Las características de las dos alternativas se representan en la tabla No. 2:

Número de Dispositivos	Arquitectura	Marca	CPU	Núcleos	RAM
1	Linux x86_64	Lenovo thinkPad W510	Intel Core i7-920XM 2.0 GHz	8	8 Gb
5	Linux x86_64	Lenovo ThinkCentre Serie M	core i7-6700T 2.8GHz	8	8 Gb
4	Android ARMv7	Samsumg Galaxy S4	Quad-core 1.9 GHz Krait 300	4	2 Gb
1	Android ARMv7	Nexus 4	Quad-core 1.5 GHz Krait,	4	2 Gb

Tabla 2. Características técnicas de los dispositivos usados para pruebas.

- **Grado de paralelismo:** Este factor indica el número de regiones en las que se dividió la imagen original y en este caso indica el número de trabajos necesarios para completar el procesamiento total de la imagen. Las dos alternativas fueron 450 y 1000 trabajos, teniendo por cada trabajo una imagen de 2Mb y 900Kb respectivamente.
- **Redundancia:** Con este factor se especifican cuantos resultados son necesarios para que un trabajo sea completado satisfactoriamente. Esto se hace con el objetivo de confirmar que un resultado es correcto al comparar resultados de la misma región procesados por diferentes clientes, usualmente es necesario cuando no se confía plenamente en los clientes. Para este factor se utilizaron los valores 1 y 2 resultados necesarios para aceptar una respuesta. Se tomaron estos dos factores para representar 2 escenarios: el primero donde las maquinas le pertenecen a la organización que genera los trabajos y confían en sus resultados y el segundo donde no se conoce la procedencia de los dispositivos y por esto se requieren al menos 2 resultados que se comparan y se consideran validos solo si son idénticos.
- **Ubicación Servidor:** En este factor se buscaba determinar el impacto de la latencia al ubicar el servidor Boinc en la red local de los dispositivos de procesamiento o en una red remota. Para el servidor remoto se utilizó un servicio provisto por AWS (Amazon Web Services) con Ubuntu 14.04, 1Gb de RAM, 1 VCPU y 10Gb de disco de estado sólido y para el servidor local se utilizó un computador con Ubuntu 14.04, 4Gb de RAM, 8 CPUs y 50Gb de disco duro.

Los factores anteriormente descritos y sus alternativas se resumen en la tabla No. 3.

Factor	Nivel -1	Nivel 1
Dispositivos, A	Computadores Linux x86_64	Dispositivos móviles Android armv7
Grado de paralelismo, B	1000	450
Redundancia, C	1	2
Ubicación Servidor, D	Local	Remoto

Tabla 3. Factores y niveles utilizados en las pruebas

2.2. Resultado de experimento 2^k

		Computadores Linux		Dispositivos móviles Android	
		1000 trabajos	450 trabajos	1000 trabajos	450 trabajos
Local	Redundancia 1	37m	15m	42m	19m
	Redundancia 2	1h 22m	38m	1h 26m	40m
Remoto	Redundancia 1	59m	48m	1h 03m	54m
	Redundancia 2	1h 54m	1h 37m	2h 01m	1h 52m

Tabla 4. Resultados de las pruebas 2^4.

Dados los resultados ilustrados en la tabla No. 4 se evaluó la importancia de cada uno de los factores, para esta prueba específica, por la proporción de variación total en la respuesta dada por cada factor. Para realizar estos cálculos se utilizó el método de tabla de signos como se puede observar en el Anexo 7: Documento de Pruebas.

La variación total está dada por la suma total de los cuadrados (SST) [89] de los resultados en la tabla y se puede expresaran en la siguiente ecuación:

$$SST = 2^4(q_A{}^2 + q_B{}^2 + q_C{}^2 + q_D{}^2 + q_{AB}{}^2 + q_{AC}{}^2 + q_{AD}{}^2 + q_{BC}{}^2 + q_{BD}{}^2 + q_{CD}{}^2 + q_{ABC}{}^2 + q_{ABD}{}^2 + q_{ACD}{}^2 + q_{BCD}{}^2 + q_{ABCD}{}^2)$$

Donde el lado derecho de la ecuación representa la porción de variación total dada por los factores A:Dispositivos, B:Grado de paralelismo, C:Redundancia y D:Ubicación del servidor, y las interacciones entre estos respectivamente. Dadas estas condiciones se encontraron los porcentajes de variación que se visualizan en la tabla No. 5:

A	0%
B	6%
C	47%
D	30%
AB	2%
AC	1%
AD	1%
BC	1%
BD	0%
CD	6%
ABC	1%
ABD	1%
ACD	2%
BCD	0%
ABCD	1%

Tabla 5. Porcentaje de variación total por factor.

A continuación, se muestran tres gráficas que ilustran los resultados del impacto de los diferentes factores en el tiempo de procesamiento. La primera, figura No. 15, muestra los resultados teniendo en cuenta los factores de tipo de dispositivos y ubicación del servidor manteniendo la redundancia en 1 y el número de trabajos en 450; también muestra el rendimiento del caso base. La segunda gráfica, Figura No. 16, compara los resultados teniendo en cuenta la redundancia y el tipo de dispositivos utilizados, manteniendo la ubicación del servidor local y el número de trabajos en 450. La ultima gráfica, Figura No. 17, muestra los resultados variando el número de trabajos y el tipo de dispositivos, manteniendo la ubicación del servidor local y la redundancia en 1.

Figura 15 Grafica tiempo de procesamiento por tipo de dispositivos y ubicación del servidor.

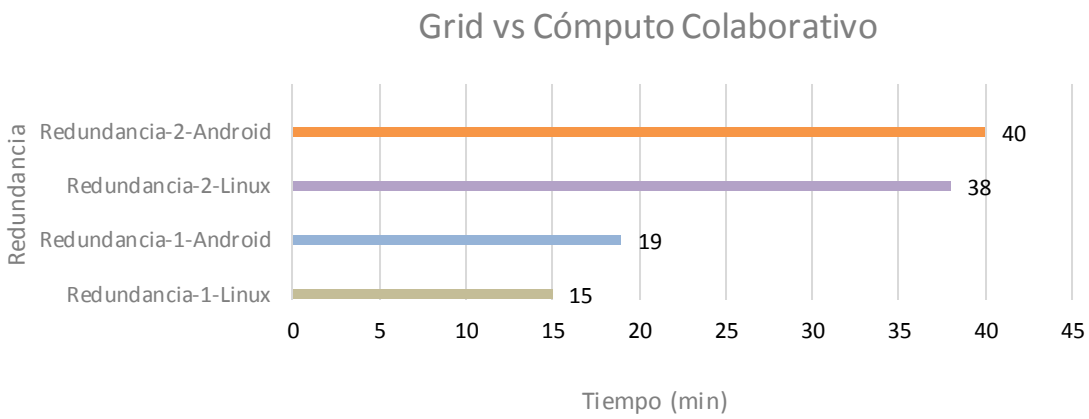

Figura 16 Grafica tipo de procesamiento con diferentes niveles de redundancia.

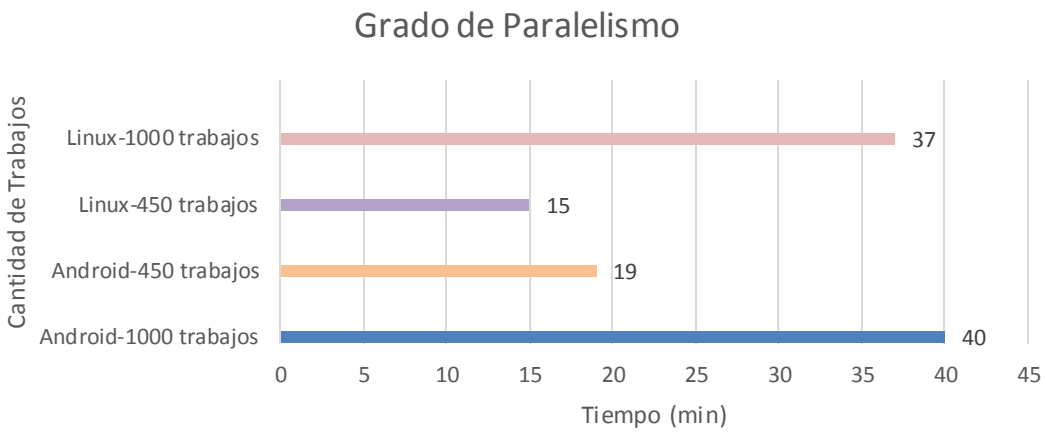

Figura 17 Grafica tiempo de procesamiento con diferente grado de paralelismo.

De los resultados obtenidos es claro que los factores que más afectan la variable respuesta son la ubicación del servidor y la redundancia de trabajos: en un 30% y 47% respectivamente, el tiempo de ejecución. La ubicación del servidor, afecta la latencia al transmitir los trabajos y los resultados; es una variable importante que se debe tener en cuenta al trabajar con esta infraestructura. Se puede concluir que idealmente los clientes deben estar en la misma red que el servidor con el fin de reducir el tiempo de transmisión de datos. Por su parte, la redundancia afecta significativamente el tiempo de ejecución pues duplica el número de trabajos que deben ser enviados y procesados en los dispositivos. Los demás factores y las interacciones entre factores no impactan de forma importante el desempeño del procesamiento.

También es importante resaltar que, al cambiar los computadores de escritorio por dispositivos móviles, el resultado se vio afectado en un porcentaje tan bajo que este factor es despreciable. En este caso se puede observar que las diferencias en las características de los dispositivos son grandes pero los clientes de Boinc utilizan un modelo no invasivo el cual no utiliza todos los recursos, de los dispositivos voluntarios al máximo, permitiendo a los usuarios a los usuarios seguir utilizando sus dispositivos sin que se vea afectado el desempeño de sus aplicaciones.

Por otro lado, la prueba base que se realizó sin utilizar un cliente Boinc, utiliza todos los recursos de la máquina, pues se observó a través del monitor del sistema el uso de todos los CPUs al 100% mientras que en los computadores, al utilizar el cliente Boinc se pudo percibir un uso de los CPUs de alrededor de 5%. Este resultado es interesante porque aun utilizando un porcentaje mucho más bajo de los recursos fue posible disminuir significativamente el tiempo de ejecución al paralelizar el procesamiento de la imagen. Este resultado también implica que con una configuración avanzada del proyecto de Boinc, que permite, por ejemplo, solicitar mayor memoria RAM o

capacidad de almacenamiento en cada cliente, puede ser posible obtener mejores resultados al utilizar más recursos de los dispositivos.

Finalmente, se observó que el grado de paralelismo, en esta prueba, afecta en un 6% el tiempo de ejecución. Es claro que este valor puede cambiar dependiendo de los niveles seleccionados al diseñar las pruebas, es por esto que, a pesar de ser bajo el porcentaje en estas pruebas, se debe tener en cuenta pues la selección adecuada de este factor, puede representar una variación significativa en el tiempo de procesamiento. Esta decisión debe ser tomada teniendo en cuenta la capacidad de los dispositivos tanto de almacenamiento como procesamiento y transmisión de datos sobre la red.

X – CONCLUSIONES

La propuesta de este Trabajo de Grado fue explorar el uso de una Grid Móvil, para el ejecutar algoritmos paralelos sencillos que procesan imágenes médicas. Se comenzó con el filtro que provee ITK: S*moothingRecursiveGaussianImageFilter* que genera una imagen difuminada realizando una convolución sobre la imagen, la cual utiliza un kernel que tiene valores dados por la distribución Gaussiana o normal [88]. La Grid Móvil se desplegó sobre Boinc como se propuso en [24].

En la búsqueda de trabajos relacionados no se encontraron experiencias similares del uso de Boinc y dispositivos móviles para procesamiento de imágenes. El trabajo con mayor relación es *Desenvolvimento de um software para análise de eletrocardiogramas utilizando dispositivos móveis* [51] donde se logra procesar, en un dispositivo móvil, un electrocardiograma e interpretar su contenido. El uso de un dispositivo es suficiente para la cantidad de procesamiento requerido en dicho trabajo.

Profundizar en este tema, poco explorado, de procesar en forma paralela imágenes en dispositivos móviles, presentaba básicamente 3 grandes retos:

- Ejecutar en dispositivos móviles, con sistema operativo Android, algoritmos que utilicen la librería ITK.
- Procesar de forma paralela imágenes de gran tamaño en múltiples dispositivos móviles.
- Conseguir que el rendimiento del procesamiento de imágenes, utilizando dispositivos móviles fuese comparable con el de computadores de escritorio.
- Evitar modificar el código del algoritmo de procesamiento de imágenes con llamadas al API de Boinc.

Con respecto al primer reto, se logró compilar ITK para Android, pero sin incluir todos los módulos del kit de herramientas. Para ello fue necesario desactivar las banderas que realizan pruebas, pues deben ejecutarse en la plataforma destino, y deshabilitar la carga de librerías en tiempo de ejecución, con el fin de generar un código ejecutable estático. Para este proyecto se construyeron el módulo central y el de filtros.

En relación al procesamiento paralelo, se logró procesar imágenes de 3101 Mb. Se desarrollaron dos componentes genéricos que facilitan la generación de trabajos y la recolección y unión de los resultados generados. Lo que provee Boinc para el procesamiento distribuido es limitado, puesto que se deben realizar extensas configuraciones manuales que exigen tener un conocimiento alto de la infraestructura, lo que lo hace complicado de manejar y poco escalable. Ahora el programador tiene a su disposición un generador de trabajos que le permite implementar su propio algoritmo de división de imagen y poder integrarlo a la infraestructura Boinc sin mayor conocimiento de su API. Asimismo, dispone de un manejador de resultados (asimilador) que simplifica el proceso de unión de resultados abstrayendo los detalles del funcionamiento de Boinc.

Con relación al desempeño, dadas las medidas preliminares, se lograron resultados muy prometedores. Usando la plataforma Boinc, el tiempo de ejecución de procesar imágenes es similar usando dispositivos móviles y computadores de escritorio. Para algunas pruebas usando un servidor local, el tiempo de ejecución del algoritmo paralelo logra mejorar el secuencial en un 14% al dividir la imagen en 1000 partes y en un 61% al dividir la imagen en 450 partes. Adicionalmente, la plataforma no invade totalmente los recursos de los dispositivos móviles de los clientes. Todas estas razones hacen interesante y atractivo el uso de Boinc como Grid Móvil para el problema de interés.

Finalmente, se logró compilar el wrapper que provee Boinc para dispositivos móviles Android y realizar este aporte al repositorio oficial del código fuente de Boinc. Gracias a esto, cualquier aplicación, a la que se le genere un ejecutable que no cargue librerías dinámicamente, puede ser enviada y ejecutada en dispositivos móviles sin tener que modificar su código.

Al haber superado los retos descritos anteriormente se concluye que los algoritmos que procesan imágenes médicas y están basados en la estrategia dividir y conquistar se pueden apoyar en la computación distribuida, y en los dispositivos móviles ociosos, utilizando una Grid Móvil.

XI – TRABAJO FUTURO

De los resultados obtenidos, se plantean los siguientes trabaos futuros.

Automatización de la creación y configuración del servidor Boinc

La configuración del servidor Boinc es un proceso largo, propenso a errores y que cambia dependiendo del sistema operativo en el que se deba montar el servidor. Se requiere que la creación de éste sea lo más rápida y transparente posible. Actualmente, hay dos trabajos que pretenden simplificar este proceso:

- El primero es una máquina virtual para VirtualBox que provee Boinc con las dependencias necesarias para crear el servidor; sin embargo, para que funcione correctamente es necesario configurar las interfaces de red y compilar y crear el servidor.
- La segunda opción es un proyecto de código abierto que crea un conjunto de contenedores usando Docker que permiten el uso de Boinc, esta opción es un poco más amigable para principiantes ya que configurar el servidor es muy fácil, con dos comandos ya funciona en cualquier computador. A pesar de esto, utilizarlo es más complicado ya que se necesita que la persona conozca de Docker para ejecutar los comandos que solicita Boinc cuando va a crear tareas o proyectos. Se necesita entonces una alternativa un poco más flexible, que facilite la creación y mantenimiento de un servidor Boinc sin que el encargado deba ser un administrador de servidores.

Interfaz gráfica de administración

El proceso de creación y configuración de aplicaciones en Boinc es manual y requiere de personal especializado. Tanto así que en la documentación oficial se recomienda 3 meses de capacitación para crear un proyecto nuevo, 1 mes de un administrador de servidores experimentado, 1 mes de un desarrollador web experimentado y 1 mes de un programador. Esta interfaz debe permitir crear proyectos y aplicaciones más fácilmente a un programador sin que conozca tanto los detalles de Boinc.

Modificación de ITK:

Actualmente no es posible hacer la compilación cruzada de la librería completa para Android. Para lograrlo es necesario modificar el proceso de construcción y el código fuente de ITK para que este problema no ocurra y se puedan utilizar todos los módulos.

Servidor Boinc en celulares

Para que Boinc funcione correctamente, es necesario que el servidor esté instalado en un computador con Ubuntu o Debian. Para la creación de una Grid ad-hoc, se pudiera explorar la posibilidad de instalar el servidor de Boinc en un dispositivo móvil.

Soporte a más plataformas

Actualmente Boinc y el wrapper de Boinc funcionan para plataformas de escritorio principalmente, es necesario para tener mayor poder de procesamiento agregar el soporte para iOS, Windows Phone, y otras plataformas de sistemas embebidos como Raspberry Pi.
En estos momentos no es posible usar Boinc en iOS ya que por políticas de Apple una aplicación no puede descargar ejecutables. Una posible solución a este problema es que todos los ejecutables necesarios ya estén dentro del cliente de Boinc.

Pruebas adicionales

A lo largo de este Trabajo de Grado se hicieron pruebas con un sólo filtro de imágenes médicas. Es necesario hacer más pruebas con un mayor número de celulares y diferentes filtros de ITK. Es importante explorar el comportamiento de otras variables respuesta que reporten el uso de los recursos de los dispositivos. También es importante hacer estas pruebas modificando las configuraciones avanzadas de Boinc tanto del cliente como del servidor y analizar los resultados. Por ejemplo, configurar el porcentaje de CPU que debe utilizar el cliente, o el tiempo máximo que debe esperar el servidor por el resultado de una tarea.

Enviar tareas utilizando un sistema Push

Actualmente los clientes de Boinc son los encargados de pedir trabajos. Cada determinado tiempo hacen una consulta al servidor por tareas nuevas. Sin embargo, para reducir este tiempo de espera se podría cambiar el modelo de envío de tareas a un modelo push. De esta manera el servidor le puede enviar al cliente una tarea en el momento que se crea, muy similar a las notificaciones push que se utilizan en dispositivos móviles.

XII- REFERENCIAS Y BIBLIOGRAFÍA

[1] A. Restrepo, «Procesamiento de imágenes médicas», *Revista Universidad EAFIT*, vol. 34, n.º 110, pp. 86–92, 2012.

[2] M. J. McAuliffe, F. M. Lalonde, D. McGarry, W. Gandler, K. Csaky, y B. L. Trus, «Medical Image Processing, Analysis and Visualization in clinical research», en *14th IEEE Symposium on Computer-Based Medical Systems, 2001. CBMS 2001. Proceedings*, 2001, pp. 381-386.

[3] V. Bharadwaj, D. Ghose, y T. G. Robertazzi, «Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems», *Cluster Computing*, vol. 6, n.º 1, pp. 7-17, ene. 2003.

[4] P. B. Hansen, «Model programs for computational science: A programming methodology for multicomputers», *Concurrency: practice and experience*, vol. 5, n.º 5, pp. 407–423, 1993.

[5] L. M. Silva y R. Buyya, «Parallel programming models and paradigms», *High Performance Cluster Computing: Architectures and Systems*, vol. 2, pp. 4–27, 1999.

[6] Yuyun Liao y D. B. Roberts, «A high-performance and low-power 32-bit multiply-accumulate unit with single-instruction-multiple-data (SIMD) feature», *IEEE Journal of Solid-State Circuits*, vol. 37, n.º 7, pp. 926-931, jul. 2002.

[7] P. Rodriguez V., «A radix-2 FFT algorithm for Modern Single Instruction Multiple Data (SIMD) architectures», 2002, p. III-3220-III-3223.

[8] D. S. Hirschberg, A. K. Chandra, y D. V. Sarwate, «Computing Connected Components on Parallel Computers», *Commun. ACM*, vol. 22, n.º 8, pp. 461–464, ago. 1979.

[9] Q. F. Stout, «Supporting divide-and-conquer algorithms for image processing», *Journal of Parallel and Distributed Computing*, vol. 4, n.º 1, pp. 95–115, 1987.

[10] I. Foster y C. Kesselman, Eds., *The grid: blueprint for a new computing infrastructure*, 2nd ed. Amsterdam ; Boston: Morgan Kaufmann, 2004.

[11] Q. Morante, N. Ranaldo, A. Vaccaro, y E. Zimeo, «Pervasive grid for large-scale power systems contingency analysis», *IEEE Transactions on Industrial Informatics*, vol. 2, n.º 3, pp. 165-175, ago. 2006.

[12] B. Jacob y International Business Machines Corporation, Eds., *Introduction to grid computing*, 1st ed. United States? IBM, International Technical Support Organization, 2005.

[13] K. B. Parmar, N. N. Jani, P. S. Shrivastav, y M. H. Patel, «Mobile Grid Computing: Facts or Fantasy?», *IJMSE*, vol. 4, n.º 1, p. 8, J ANUARY 2013.

[14] «En Colombia hay 55 millones de líneas de telefonía móvil», *ElEspectador*, 30-mar-2015. [En línea]. Disponible en: http://www.elespectador.com/noticias/economia/colombia-hay-55-millones-de-lineas-de-telefonia-movil-articulo-552382. [Accedido: 19-may-2016].

[15] «Colombia, el país de los 'smartphones'». [En línea]. Disponible en: http://www.semana.com/tecnologia/articulo/colombia-el-pais-de-los-smartphones/432806-3. [Accedido: 19-may-2016].

[16] S. P. Ahuja y J. R. Myers, «A survey on wireless grid computing», *The Journal of Supercomputing*, vol. 37, n.º 1, pp. 3–21, 2006.

[17] J. M. Rodriguez, A. Zunino, y M. Campo, «Introducing mobile devices into Grid systems: a survey», *International Journal of Web and Grid Services*, vol. 7, n.º 1, pp. 1-40, ene. 2011.

[18] N. Palmer, R. Kemp, T. Kielmann, y H. Bal, «Ibis for mobility: solving challenges of mobile computing using grid techniques», 2009, pp. 1-6.

[19] L. dos S. Lima, A. T. A. Gomes, A. Ziviani, M. Endler, L. F. G. Soares, y B. Schulze, «Peer-to-peer resource discovery in mobile Grids», 2005, pp. 1-6.

[20] J. M. Jaehnert, S. Wesner, y V. A. Villagra, *The akogrimo mobile grid reference architecture-overview*. Citeseer.

[21] A. Wolff, S. Michaelis, J. Schmutzler, y C. Wietfeld, «Network-centric Middleware for Service Oriented Architectures across Heterogeneous Embedded Systems», 2007, pp. 105-108.

[22] A. Coronato y G. De Pietro, «MiPeG: A middleware infrastructure for pervasive grids», *Future Generation Computer Systems*, vol. 24, n.º 1, pp. 17-29, ene. 2008.

[23] A. Litke, D. Halkos, K. Tserpes, D. Kyriazis, y T. Varvarigou, «Fault tolerant and prioritized scheduling in OGSA-based mobile Grids», *Concurrency Computat.: Pract. Exper.*, vol. 21, n.º 4, pp. 533-556, mar. 2009.

[24] R. R. García, «Grids Accesibles», Pontificia Universidad Javeriana.

[25] D. P. Anderson, «BOINC: a system for public-resource computing and storage», en *Fifth IEEE/ACM International Workshop on Grid Computing, 2004. Proceedings*, 2004, pp. 4-10.

[26] National Library of Medicine, «ITK - Segmentation & Registration Toolkit». [En línea]. Disponible en: https://itk.org/. [Accedido: 20-may-2016].

[27] L. Ibanez, W. Schroeder, L. Ng, y J. Cates, «The ITK software guide», 2003.

[28] k R Sriraman, «Grid Computing on Mobile Devices». ALTIMETRIK.

[29] H. Ba, W. Heinzelman, C.-A. Janssen, y J. Shi, «Mobile computing - A green computing resource», en *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, 2013, pp. 4451-4456.

[30] The Process Group, «DECISION ANALYSIS AND RESOLUTION PROCESS», *POST*, vol. 12, No. 2, oct-2005.

[31] I. Foster, «The Anatomy of the Grid: Enabling Scalable Virtual Organizations», *International Journal of High Performance Computing Applications*, vol. 15, n.º 3, pp. 200-222, ago. 2001.

[32] W. Y. Zeng, Y. L. Zhao, J. W. Zeng, y W. Song, «Mobile Grid Architecture Design and Application», en *2008 4th International Conference on Wireless Communications, Networking and Mobile Computing*, 2008, pp. 1-4.

[33] D. P. Anderson, «Volunteer computing: the ultimate cloud.», *ACM Crossroads*, vol. 16, n.º 3, pp. 7-10, 2010.

[34] «VolunteerComputing – BOINC». [En línea]. Disponible en: https://boinc.berkeley.edu/trac/wiki/VolunteerComputing. [Accedido: 16-nov-2016].

[35] University of California, «BOINC», *Boinc Open-source software for volunteer computing*, 2016. [En línea]. Disponible en: http://boinc.berkeley.edu/. [Accedido: 19-may-2016].

[36] P. A. Bernstein, «Middleware: a model for distributed system services», *Communications of the ACM*, vol. 39, n.º 2, pp. 86-98, feb. 1996.

[37] D. Mackenzie y T. Tromey, «GNU Automake Manual», *Free Software Foundation*, 2003.

[38] C. P. Walters y J. A. Brown, *Dynamic cross-compilation system and method*. Google Patents, 1998.

[39] S. Pratschner, «Simplifying deployment and solving DLL Hell with the .NET framework», *MSDN Magazine*, 2001.

[40] D. A. Wheeler, «Program library howto», *Online, April*, p. 27, 2003.

[41] A. E. De Giusti *et al.*, «Algoritmos paralelos sobre arquitecturas multicluster y GRID», en *IX Workshop de Investigadores en Ciencias de la Computación*, 2007.

[42] C. A. Glasbey y G. W. Horgan, *Image analysis for the biological sciences*. J. Wiley, 1995.

[43] Robert Fisher, Simon Perkins, Ashley Walker, y Erik Wolfart, «IMAGE PROCESSING LEARNING RESOURCES HIPR2». [En línea]. Disponible en: http://homepages.inf.ed.ac.uk/rbf/HIPR2/hipr_top.htm. [Accedido: 16-oct-2016].

[44] D. West, «How mobile devices are transforming healthcare», *Issues in technology innovation*, vol. 18, n.º 1, pp. 1–11, 2012.

[45] M. Boulos, S. Wheeler, C. Tavares, y R. Jones, «How smartphones are changing the face of mobile and participatory healthcare: an overview, with example from eCAALYX», *BioMedical Engineering OnLine*, vol. 10, n.º 1, p. 24, 2011.

[46] «OGSI.net Main page». [En línea]. Disponible en: http://www.cs.virginia.edu/~gsw2c/ogsi.net.html. [Accedido: 09-may-2016].

[47] D. C. Chu y M. Humphrey, «Mobile OGSI.NET: Grid Computing on Mobile Devices», en *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, Washington, DC, USA, 2004, pp. 182–191.

[48] M. Alia, S. Hallsteinsen, N. Paspallis, y F. Eliassen, «Managing Distributed Adaptation of Mobile Applications», en *Distributed Applications and Interoperable Systems*, J. Indulska y K. Raymond, Eds. Springer Berlin Heidelberg, 2007, pp. 104-118.

[49] A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikainen, y V. H. Tuulos, «Using mapreduce framework for mobile applications», *Multimedia Services and Streaming for Mobile Devices: Challenges and Innovation*, vol. 181, 2011.

[50] J. Dean y S. Ghemawat, «MapReduce: simplified data processing on large clusters», *Communications of the ACM*, vol. 51, n.º 1, pp. 107–113, 2008.

[51] A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikainen, y V. H. Tuulos, «Misco: A MapReduce Framework for Mobile Systems», en *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*, New York, NY, USA, 2010, p. 32:1–32:8.

[52] Austin Davis y Ronald Lai, «Elastic Parallel Execution Framework on Android», *15-418 S14: Parallel Computer Architecture and Programming*, 2014. [En línea]. Disponible en: https://www.contrib.andrew.cmu.edu/~awdavis/15418/project/. [Accedido: 19-may-2016].

[53] J. P. Folador, «Desenvolvimento de um software para análise de eletrocardiogramas utilizando dispositivos móveis», Programa de Mestrado Profissional em Inovação Tecnológica, Universidade Federal do Triângulo Mineiro, Uberaba, 2015.

[54] «IBM Press | Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise», 12-jun-2012. [En línea]. Disponible en: https://www.redbooks.ibm.com/Redbooks.nsf/ibmpressisbn/9780132810135. [Accedido: 19-may-2016].

[55] «Disciplined Agile 2.0 A Process Decision Framework for Enterprise I.T.», *Disciplined Agile 2.0*, 08-abr-2014. .

[56] G. R. A. 13 y 2016 at 7:51 Am, «A Hybrid Framework», *Disciplined Agile 2.X*, 27-abr-2014. .

[57] «Introduction to DAD», *Disciplined Agile 2.X*, 11-nov-2012. .

[58] «Full Agile Delivery Lifecycles», *Disciplined Agile 2.0*, 06-may-2014. .

[59] S. Ambler, «An Exploratory "Lean Startup" Lifecycle», *Disciplined Agile 2.X*, 25-abr-2014. .

[60] B. Phifer, «DAR Basics: Applying Decision Analysis and Resolution in the Real World», *Software Engineering Institute, Carnegie Mellon University, SEPG Presentation, http://www.sei.cmu.edu/cmmi/presentations/sepg04.presentations/dar.pdf, USA*, p. 32, 2004.

[61] «BoincIntro – BOINC». [En línea]. Disponible en: https://boinc.berkeley.edu/trac/wiki/BoincIntro. [Accedido: 17-oct-2016].

[62] «gnu.org». [En línea]. Disponible en: https://www.gnu.org/copyleft/lesser.html. [Accedido: 17-oct-2016].

[63] «AppVersionNew – BOINC». [En línea]. Disponible en: https://boinc.berkeley.edu/trac/wiki/AppVersionNew. [Accedido: 28-oct-2016].

[64] «ServerComponents – BOINC». [En línea]. Disponible en: https://boinc.berkeley.edu/trac/wiki/ServerComponents. [Accedido: 17-oct-2016].

[65] «MasterUrl – BOINC». [En línea]. Disponible en: https://boinc.berkeley.edu/trac/wiki/MasterUrl. [Accedido: 17-oct-2016].

[66] «BasicConcepts – BOINC». [En línea]. Disponible en: https://boinc.berkeley.edu/trac/wiki/BasicConcepts. [Accedido: 17-oct-2016].

[67] «BoincPlatforms – BOINC». [En línea]. Disponible en: https://boinc.berkeley.edu/trac/wiki/BoincPlatforms. [Accedido: 27-oct-2016].

[68] «JobIn – BOINC». [En línea]. Disponible en: https://boinc.berkeley.edu/trac/wiki/JobIn. [Accedido: 17-oct-2016].

[69] «BackendPrograms – BOINC». [En línea]. Disponible en: https://boinc.berkeley.edu/trac/wiki/BackendPrograms. [Accedido: 27-oct-2016].

[70] «ValidationIntro – BOINC». [En línea]. Disponible en: http://boinc.berkeley.edu/trac/wiki/ValidationIntro. [Accedido: 27-oct-2016].

[71] «ValidationSimple – BOINC». [En línea]. Disponible en: http://boinc.berkeley.edu/trac/wiki/ValidationSimple. [Accedido: 27-oct-2016].

[72] «AssimilateIntro – BOINC». [En línea]. Disponible en: https://boinc.berkeley.edu/trac/wiki/AssimilateIntro. [Accedido: 27-oct-2016].

[73] «DataFlow – BOINC». [En línea]. Disponible en: http://boinc.berkeley.edu/trac/wiki/DataFlow. [Accedido: 17-oct-2016].

[74] «HostMeasurement – BOINC». [En línea]. Disponible en: https://boinc.berkeley.edu/trac/wiki/HostMeasurement. [Accedido: 13-nov-2016].

[75] «AppIntro – BOINC». [En línea]. Disponible en: https://boinc.berkeley.edu/trac/wiki/AppIntro. [Accedido: 17-oct-2016].

[76] «DigitalOcean: Cloud computing designed for developers», *DigitalOcean*. [En línea]. Disponible en: https://www.digitalocean.com/. [Accedido: 17-oct-2016].

[77] University of California, «The BOINC Wrapper», *Boinc Open-source software for volunteer computing*, 2014. [En línea]. Disponible en: http://boinc.berkeley.edu/trac/wiki/WrapperApp. [Accedido: 19-may-2016].

[78] «ExampleApps – BOINC». [En línea]. Disponible en: http://boinc.berkeley.edu/trac/wiki/ExampleApps. [Accedido: 17-oct-2016].

[79] «AndroidBuildApp – BOINC». [En línea]. Disponible en: http://boinc.berkeley.edu/trac/wiki/AndroidBuildApp. [Accedido: 18-oct-2016].

[80] «Security Enhancements in Android 5.0 | Android Open Source Project». [En línea]. Disponible en: https://source.android.com/security/enhancements/enhancements50.html. [Accedido: 22-oct-2016].

[81] H. Shewale, S. Patil, V. Deshmukh, y P. Singh, «Analysis of android vulnerabilities and modern exploitation techniques», *ICTACT Journal on Communication Technology*, vol. 5, n.º 1, 2014.

[82] «[FIX] [Android "L"] Bypassing the new PIE security check», *XDA Developers*. [En línea]. Disponible en: http://forum.xda-developers.com/google-nexus-5/development/fix-bypassing-pie-security-check-t2797731. [Accedido: 18-oct-2016].

[83] H. Bojinov, D. Boneh, R. Cannings, y I. Malchev, «Address space randomization for mobile devices», 2011, p. 127.

[84] S. Liebergeld y M. Lange, «Android Security, Pitfalls and Lessons Learned», en *Information Sciences and Systems 2013*, E. Gelenbe y R. Lent, Eds. Springer International Publishing, 2013, pp. 409-417.

[85] «Android versions market share 2016», *Statista*. [En línea]. Disponible en: https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/. [Accedido: 10-nov-2016].

[86] J. J. Caban, A. Joshi, y P. Nagy, «Rapid development of medical imaging tools with open-source libraries», *Journal of Digital Imaging*, vol. 20, n.º 1, pp. 83–93, 2007.

[87] «CMake». .

[88] D. Hale, «Recursive gaussian filters», *CWP-546*, 2006.

[89] P. N. D. Bukh y R. Jain, *The art of computer systems performance analysis, techniques for experimental design, measurement, simulation and modeling*. JSTOR, 1992.

XIII - ANEXOS

Anexo 1. Tutorial para crear un proyecto Boinc.

Anexo 2. Configurar Servidor Boinc.

Anexo 3. Requerimientos del sistema.

Anexo 4. Compilar Wrapper Para Android.

Anexo 5. Compilación Cruzada de ITK.

Anexo 6. Documento de Diseño de Software SDD.

Anexo 7. Documento de Pruebas.

Anexo 8. Compilación cruzada de aplicación ITK.