APROVECHAMIENTO DE CIBER INFRAESTRUCTURAS PARA DESARROLLAR Y EJECUTAR ALGORITMOS E-SCIENCE POR REDUCCIÓN DE DATOS QUE REQUIEREN DE HPC EN LA UTB

AUTOR: MANUEL GUILLERMO CASTRO ARIZA

TESIS DE GRADO PARA OPTAR POR EL MAGISTER EN INGENIERÍA

DIRECTOR: MSc. JAIRO SERRANO

INGENIERO DE SISTEMAS

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR

FACULTAD DE INGENIERÍA

MAESTRÍA EN INGENIERÍA CON ÉNFASIS EN SISTEMAS

CARTAGENA, COLOMBIA

2015

Dedicatoria

Dedico este momento a Ma, Pa, Danny, Andre, Abues, Toñi y mi Tía, por su patrocinio, y apoyo moral, emocional y espiritual, durante estos dos años que tomó finalizar este trabajo, y quienes estuvieron gran parte de este tiempo dando sus aportes y consejos, los cuales considero de incuantificable importancia. Dedicatoria muy especial para mi novia Tati León, un soporte a nivel moral, emocional y académico de otro nivel.

Agradecimientos

Agradecimientos especiales a mi tutor Jairo Serrano, quien expuso ante mí la temática que trató este trabajo, y quien guió constantemente en su desarrollo. A mi amigo y coequipero José Bolaños, darle muchas gracias por sus consejos, ayudas y observaciones constructivas a lo largo de este proceso académico. Asimismo, dar enormes gracias a profesores Luz Stella Robles, José Luis Villa, William Caicedo, Julio Seferino Hurtado, Edgardo Arrieta, Juan Carlos Martínez, Issac Zuñiga, Martin Monroy y David Franco por los sustanciales y oportunos aportes en conocimientos, guías, y por su amabilidad.

Reconocimiento

Muchas gracias a la UTB y a sus estudiantes e ingenieros de sistemas, por permitirme ser parte de su núcleo, y darme oportunidad de intentar brindar un pequeño aporte a su comunidad académica.

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE GRÁFICAS

LISTA DE CÓDIGOS

LISTA DE ANEXOS

TABLA DE CONTENIDO

GLOSARIO

GCC (GNU COMPILER COLLECTION)

Colección de sistemas de compilación para múltiples lenguajes de programación. Estas herramientas permiten desarrollar, probar y compilar código de lenguajes como C, C++, Fortran, Objective-C, Java y otros en variadas arquitecturas de procesamiento y sistemas operativos [65].

HTCONDOR

Conocido popularmente como Condor, este planificador de tareas gestiona la ejecución de las mismas a través de un sistema de computación distribuido. HTCondor les brinda varias interfaces a sus usuarios con motivo de que estos puedan hacer uso de diferentes plataformas de computación distribuida [70].

MÁQUINAS VIRTUALES

Son sistemas emuladores de otros sistemas. Generalmente se emplean para emular sistemas operativos que operan sobre sistemas operativos “reales”. Entre los usos típicos que se le dan a estos sistemas se encuentra: redundancia para protección contra fallos y reproducción de entornos Linux para gestionar servicios.Existe software gestor de máquinas virtuales tales como VMWare, Virtualbox, Vagrant, entre otros [66].

MONITOREO DE RED

El uso de sistemas para comprobar el comportamiento de una red de datos se conoce como monitoreo de red. En cualquier negocio o implementación que requiere de red de datos es necesario establecer el estado de enlaces, capacidad de tráfico, uso de ancho de banda, fallas, entre otros. Existen herramientas de monitoreo de red tales como, Munin, Wireshark, Microsoft Network Monitor, Nagios, y otros [68].

NETWORK FILE SYSTEM (NFS)

Es un protocolo de sistema de almacenamiento que permite compartir archivos de forma centralizada. El protocolo se deriva del sistema Open Network Computing Remote Procedure Call, con lo cual es abierto a que cualquier persona u organización lo implemente en sus sistemas [67].

RED HAT ENTERPRISE LINUX (RHEL)

Red Hat Enterprise Linux es una distribución comercial del sistema operativo Linux. De los componentes no comerciales de esta distribución se apoyan otras distribuciones Linux gratuitas, mantenidas por diferentes comunidades. Entre las distribuciones gratuitas basadas en RHEL se encuentran CentOS, Fedora, Scientific Linux, y otras [64].

SECURE SHELL (SSH)

Este protocolo de red encripta comunicaciones remotas de datos por línea de comandos. Es un protocolo comúnmente empleado para acceder a servicios remotos en sistemas operativos tipo Unix, p. ej. Linux [69].

RESUMEN

La Universidad Tecnológica de Bolívar, (Colombia), hace uso de un servidor/supercomputador HPC, que es utilizado por múltiples académicos de la institución para ejecutar sus algoritmos e-Science. Dicha ciber infraestructura HPC, no suple la demanda requerida por sus usuarios, generando retrasos en las actividades HPC de estos. La institución cuenta con otras ciber infraestructuras, como laboratorios de cómputo y servidores multipropósito, las cuales pueden ser mejor aprovechadas para suplir la demanda requerida de ciber infraestructuras de ejecución de tareas HPC. Adicionalmente, la ciber infraestructura HPC puede ser mejor explotada si se cuenta con herramientas software que amplíen el abanico de alternativas y faciliten el desarrollo de los algoritmos que van a ejecutarse en estas nuevas configuraciones HPC.

Con base en lo anterior, se planteó una labor investigativa con la que se despliega un entorno de ejecución HPC acorde con lineamientos de una arquitectura de sistema definida que permita mejor aprovechamiento de ciber infraestructuras existentes en la UTB. Adicionalmente, se desarrollaron bases de un framework software de paralelización, inspirado en concepto de esqueletos algorítmicos que busca ser alternativa y facilitar las labores de desarrollo de los usuarios del nuevo entorno HPC. Finalmente, se concluye el trabajo con la cuantificación de métricas de complejidad que permiten determinar si dicho framework proporciona mayor facilidad en el desarrollo de un algoritmo e-Science, contra utilizar MPI puro como herramienta de desarrollo.

Palabras claves: Ciber infraestuctura, HPC, e-Science, aprovechamiento, cluster, grid, MPI, esqueletos algorítmicos, framework, métricas de complejidad, esfuerzo de programación.

CAPÍTULO I: INTRODUCCIÓN

El profesorado y alumnado de la Universidad Tecnológica de Bolívar que emplea el servidor/supercomputador de la institución para ejecutar algoritmos e-Science, en repetidas ocasiones se ve obligado a suspender o posponer las mismas porque tal recurso esta en uso por otros usuarios de la institución. Mientras esto sucede, también se presenta en ciertos momentos la situación de que algún(os) laboratorio(s) de cómputo están siendo subutilizados, ya sea porque no se emplean para realizar prácticas académicas o no se explota toda la capacidad computacional que estos pueden proporcionar.

Otras instituciones académicas se han visto enfrentadas a problemática similares, y han capitalizado la oportunidad que representa el aprovechar estos recursos computacionales o ciber infraestructuras de mejor forma [3] [4] [6] [27]. Tal aprovechamiento, se materializó a través del despliegue de redes de cómputo, locales y amplias, de alta capacidad de procesamiento y memoria, e integración de herramientas software que brinden alternativas y facilidades, para que sus usuarios logren explotar los servicios HPC con el objetivo de desarrollar y ejecutar sus tareas e-Science.

Un aspecto que se destaca en el marco de la problemática del bajo aprovechamiento de ciber infraestructuras para HPC, son los medios que los usuarios utilizan para desarrollar sus algoritmos [17], los cuales por la naturaleza misma de las ciber infraestructura, son algoritmos distribuidos, lo que generalmente dificulta su implementación por parte del usuario. En respuesta a esto, una importante cantidad de investigadores se han dado a la tarea de crear, proponer, implementar y distribuir herramientas software que satisfaga esta condición [19].

Con base en lo mencionado, en este texto se exponen los resultados de una labor investigativa, cuya principal meta es generar un ecosistema de aprovechamiento de

ciber infraestructuras HPC en la Universidad Tecnológica de Bolívar para desarrollar y ejecutar algoritmos e-Science. Tal labor comprende de actividades como el diseño, implementación y despliegue de ciber infraestructura hardware HPC para ejecutar algoritmos; diseño, implementación, pruebas, despliegue y valoración de ciber infraestructura software que posibilite y facilite el desarrollo de estos algoritmos.

La disposición de los siguientes capítulos de este trabajo de investigación se muestra a continuación:

Capítulo II: El Problema. En este capítulo se presenta la problemática que aborda este trabajo, se expone el repaso de la literatura respecto a lo que otros autores, instituciones y organizaciones realizan para afrontar la problemática o problemáticas similares. También se especifican los objetivos específicos y el general que este trabajo busca ejecutar exitosamente, y adicionalmente se exhibe tanto la hipótesis planteada, como la pregunta de investigación.

Capítulo III: Marco Teórico. Este capítulo describe los conceptos y teorías que este trabajo utiliza para diseñar un plan metodológico, que luego se logre materializar mediante ejecución de actividades que busquen satisfacer los objetivos del trabajo. Se relacionan conceptos sobre e-Science, HPC, ciberinfraestructuras, paradigma de programación paralela, concepto de abstracción y métricas de complejidad

Capítulo IV: Metodología. En el capítulo de la metodología se exponen criterios y argumentos con los cuales se busca satisfacer el objetivo general de este trabajo de investigación. La hoja de ruta trazada por las actividades de los objetivos específicos es la que lleva a ella. Se argumenta sobre actividades que posibilitan despliegue de entorno de ejecución para algoritmos HPC, producción de bases framework de paralelización inspirado en concepto de esqueletos algorítmicos, y finalmente sobre la evaluación de que tan provechoso es para un usuario utilizar dicho framework al momento de implementar un algoritmo, respecto a utilizar MPI puro directamente.

Capítulo V: Presentación y Análisis de Resultados. Con base en las salidas producidas en cada actividad de los diferentes objetivos específicos de este trabajo, se exponen análisis sobre los mismos de forma detallada.

Capítulo VI: Conclusiones. Se finaliza el escrito investigativo mostrando conclusiones finales en referencia a los análisis de las salidas producidas en las diferentes actividades del trabajo. Adicionalmente, se identifican oportunidades de mejora con referencia a los resultados actuales y se realizan planteamientos para extender y complementar este trabajo en un futuro.

CAPÍTULO II: EL PROBLEMA

2.1. PLANTEAMIENTO DEL PROBLEMA

La Universidad Tecnológica de Bolívar (UTB) provee soluciones HPC a sus investigadores mediante un servidor/supercomputador con procesador de 32 núcleos y 256 GB de memoria principal. Dicho supercomputador no proporciona la disponibilidad suficiente para la ejecución de tareas e-Science que requieren de HPC, lo que retrasa ejecución de actividades e-Science de sus usuarios; y la adquisición de más equipos de esta naturaleza puede costar fácilmente cientos de miles de dólares, opción que no es viable a corto plazo para la institución. La baja disponibilidad de este recurso HPC se evidenció durante sesiones de clases y momentos donde algunos docentes intentaban ejecutar algoritmos HPC, sin embargo esto debía retrasarse como consecuencia de que otro algoritmo estaba ejecutándose en el servidor HPC por tiempo prolongado. Adicionalmente, existe ciber infraestructura en la universidad que puede ser mejor aprovechada en materia de HPC, a nivel hardware, como lo son sus servidores multipropósito y laboratorios de cómputo. Finalmente, a nivel ciber infraestructura de desarrollo, se identifica que desarrollar algoritmos que requieren de HPC no es tarea sencilla para aquellos que no son versados en las tecnologías asociadas, por lo que no se aprovechan las existentes para crear nuevas ciber infraestructuras que puedan aliviar esta situación.

En la subsección a continuación, se hace repaso de la literatura respecto a problemática de bajo grado de aprovechamiento de ciber infraestructura para desarrollo y ejecución de algoritmos e-Science que requieren de HPC.

2.2. ESTADO DEL ARTE

Las ciencias computacionales desde sus inicios son factor clave en la contribución al desarrollo y generación de nuevo conocimiento en diversas áreas de la sociedad,

desde la industria aeronáutica, autos y redes, hasta en temas de protección y prevención medioambiental, medicina, las mismas ciencias, sectores energéticos [1], sólo por enumerar algunos. Es tal su aporte al descubrimiento de nuevo conocimiento, que inclusive, se le determina como el tercer pilar de la ciencia por detrás de la teoría y la experimentación [2]. La computación hoy en día es un acompañante fundamental, sino necesario, para cualquier investigador que desee llevar a cabo un proceso investigativo.

Una de las contribuciones más significativas de las ciencias computacionales a las demás ciencias es la ciber infraestructura. Cuando se habla de ciber infraestructura, se habla de artefactos físicos y abstractos de los que estudiosos de las ciencias computacionales, ingenieros de sistemas, entre otros, hacen uso diariamente en sus labores, van desde elementos físicos como servidores, cables de red, enrutadores, monitores, procesadores, chips de memoria y discos en estado sólido, hasta artefactos abstractos en mayor o menor grado, tales como bases de datos, lenguajes de programación, paradigmas de programación, aplicaciones, entre otras. Con estas herramientas los investigadores apoyan sus teorías y experimentos de manera efectiva [54].

A raíz de la notable evolución tecnológica en la ciber infraestructura, las ciencias toman provecho de dichos recursos en aras de maximizar los resultados de sus tareas investigativas. Tal uso de la ciber infraestructura es lo que origina la rama de la ciencia hoy conocida como e-Science, de electronic-Science o enhanced Science [2]. La e-Science es practicada por gran cantidad de investigadores alrededor del planeta, y uno de sus principales centros de acción son las instituciones académicas con énfasis en la investigación. Estas instituciones lideradas por colosos como MIT y UCB, por nombrar dos, preparan el terreno en cuanto a la práctica de e-Science, ejemplificada en proyectos masivos tales como ATHENA y BOINC respectivamente. Muchas instituciones educativas siguen los pasos de estos y otros gigantes de la academia y capitalizan sobre la ciber infraestructura; por nombrar algunas: Politécnico de Turín en Italia [4], universidad de los Andes [3] y Tecnológica de Bolívar, al aterrizar en el plano local/regional, entre otras.

Una manera de garantizar el aprovechamiento de ciber infraestructura de cualquier índole es mediante la implementación de mecanismos de escalabilidad. La escalabilidad en computación permite incrementar/reducir funcionalidad y rendimiento en un sistema, al promover alta reusabilidad de forma relativamente sencilla. Por ejemplo, si se desea implementar un software que permita representar por pantalla figuras geométricas básicas como el círculo, los cuadrados y triángulos, pero que a medida que pase el tiempo se desea que el software pueda representar figuras cómo la elipse, el hexágono y la estrella de seis puntas, este entonces debe proporcionar interfaces e implementación de técnicas de herencia que posibiliten que a partir de las propiedades y comportamientos de las figuras básicas, se puedan reproducir las figuras más complejas de manera eficiente, tanto para los programadores, como para el sistema que ejecuta la aplicación. Este es un pequeño ejemplo a nivel software de como la escalabilidad permite aprovechar funcionalidades existentes de un programa para crear otras.

Si se realiza una analogía con el ejemplo anterior, algo similar se puede aplicar con la ciber infraestructura a nivel hardware. Una alternativa muy usada para aprovechar recursos físicos para HPC, entre otras, es la escalabilidad, lograda mediante el uso de clusters y grids [5]. Los clusters y grids son sinónimos de escalabilidad porque son redes tipo LAN y WAN respectivamente, y de por si las redes de computadores son diseñadas con el objetivo de que se le puedan añadir o remover nodos, quienes en este caso particular serían computadoras que proporcionan procesamiento para emular un gran supercomputador.

La Ph. D. Claudia Jiménez, en un artículo publicado en la revista Sistemas de la Asociación Colombiana de Ingenieros de Sistemas (ACIS), expone como la grid permite el procesamiento y almacenamiento masivo de datos. Ella apunta al proyecto científico DataGrid de CERN como uno de los primeros esfuerzos masivos de HPC través de la grid [5]. Esta grid consiste en la interconexión de múltiples clusters (redes de computadores locales para HPC), e inclusive otras grids, distribuidas a lo largo de centros de investigaciones en territorio europeo con el objetivo de ofrecer servicios de almacenamiento y manipulación de datos para sus afiliados [5]. DataGrid es operada por medio del middleware Globus de la OGSA (Open Grid Software Architecture). Entre las tareas investigativas sometidas a DataGrid se encuentran proyectos de biología, construcción de imágenes médicas

y fotos satelitales. Esta grid evoluciona con el tiempo en EGEE (Enabling Grids for E-science in Europe), constituida por alrededor de 32 países europeos, adicionalmente, Jiménez menciona sobre iniciativas similares como EELA, una alianza europea-latinoamericana para el aprovechamiento de la ciber infraestructura, y TeraGrid en Norteamérica [5], lo cual reafirma la viabilidad de este tipo de ciber infraestructuras escalables.

A continuación, se examina las experiencias de varias instituciones con énfasis en la investigación, al momento de aprovechar su ciber infraestructura mediante escalación con clusters y grids, con el objetivo de proveer procesamiento HPC para tareas e-Science que la demanden.

Miembros del departamento de ciencias de la computación de la universidad de Purdue en Indiana describen en un artículo publicado en IEEE Xplore en 2013 como CRIS (Computational Research Infrastructure for Science) fue implementado en la institución en búsqueda de facilitar las labores científicas de sus investigadores. Los autores del texto expresan que los científicos tradicionalmente manejan ciber infraestructura y metodologías ad-hoc debido a su poca pericia en el tema. En consecuencia, su labor científica se entorpecía, era ineficiente, limitaba la capacidad de apoyarse en el trabajo de colegas y limitaba la extensibilidad de las investigaciones por parte de otras comunidades científicas. En respuesta a esto, la institución apoyó el desarrollo de ciber infraestructuras para la administración y gestión de datos científicos, que fueran fáciles de usar, escalables y colaborativas [6].

Por otro lado, se encuentra que el Politécnico de Turín goza de una situación similar de aprovechamiento de ciber infraestructura. El departamento de informática del politécnico comenzó en el año 2008 un proyecto denominado CASPER (cluster Appliance for Scientific Parallel Execution and Rendering), cuyo objetivo inicial era contribuir a las labores investigativas de la institución por medio de clusters HPC. El proyecto empieza a producir resultados traducidos en artículos de investigación y uso sostenido de la ciber infraestructura en 2011, lo que motiva a su expansión, lo que se tradujo en un nuevo nombre: HPC@POLITO, y nuevas especificaciones

técnicas: 136 núcleos para procesamiento, 584 GB de memoria principal para contribuir a un rendimiento pico de 1.2 TFLOPS [4].

En territorio colombiano la Universidad de los Andes aplica aprovechamiento de ciber infraestructuras, pero no sólo se remite al hecho de implementar estructuras escalables sino también de carácter oportunista. Por clusters o grids oportunistas Castro, Villamizar, Rosales y Giménez las señalan como aquellas que aprovechan ciber infraestructura existente, computadores de escritorio subutilizados por ejemplo, para incrementar, escalar el poder de procesamiento del sistema distribuido, y que en definitiva contribuye a ejecutar tareas e-Science. Los autores con esto describen el proyecto UnaGrid [3].

UnaGrid es una grid oportunista que utiliza computadoras de laboratorios de informática de la Universidad de los Andes para procesar tareas e-Science de la institución. La grid se compone de clusters virtualizados en múltiples computadores de la universidad e interconectados por computadores maestros que emplean middleware Globus para gestionar los recursos computacionales y tareas del sistema. La arquitectura del sistema constaba de tres clusters, uno por laboratorio de informática y tres computadores maestros que gestionan la grid; el sistema cuenta con acceso remoto por página web, con respectivas funcionalidades de monitoreo y gestión de tareas. La ventaja de este sistema distribuido oportunista es que utilizó procesamiento ocioso de computadores que por ejemplo usan estudiantes sin que estos noten deterioro en rendimiento, benefician labores investigativas, y ahorra costos a la hora de invertir en sistemas HPC dedicados [3].

Los resultados de utilizar este tipo de ciber infraestructura oportunista en la Universidad de los Andes son positivos. Se realizaronn dos pruebas de impacto de rendimiento en los computadores de los laboratorios, una donde el usuario realizaba tareas de alta demanda para el procesador y otra donde el usuario realizaba funciones de entrada y salida. En ambos caso no se evidenció mayor pérdida de desempeño, ya que las máquinas virtuales que gestionan los recursos computacionales sólo usaban aquellos en estado ocioso [3].

De forma similar a UniAndes, en la Universidad de Huddersfield se implementa un esquema de clusters y grid oportunista empleando ‘pools’ con tecnología HTCondor, ello para aprovechar los recursos computacionales subutilizados por los laboratorios de cómputo en horas que no se dictan clases. La experiencia con Condor es extremadamente positiva en este caso, y no solo se logra sacar provecho de los laboratorios a nivel de ejecución de tareas e-Science, sino que adicionalmente el middleware se adapta a una estrategia de ‘Green IT’, que permite que a bajo consumo de energía por parte de los computadores de dichas salas, estos logren ejecuten los algoritmos que envían los usuarios a los clusters o grids [27].

Mientras tanto, investigadores de la Universidad de Suceava en Rumania, resaltan como el uso de ciber infraestructura hardware y software les permite disponer de estas para proponer entornos HPC en el oriente Europeo, que posibilite la satisfacción de necesidades investigativas de los mismos. Como tarea incial, los investigadores rumanos realizaron un estudio preliminar de las capacidades HPC del cluster local de la universidad, por ejemplo, para determinar cómo este puede satisfacer necesidades de video transmisión en alta definición, entre varios centros investigativos en la región [28].

Por otro lado, en Estados Unidos existen instituciones que emplean clusters con procesamiento por tarjeta de video o GPU. Las GPU (Graphics Processing Unit) son chips dedicados cuyo propósito inicial era el de ejecutar instrucciones computacionales relacionadas con manipulación de datos para generar imágenes. Estos chips tradicionalmente vienen en todos los computadores personales y móviles, y son empleados exclusivamente para acelerar las representaciones visuales que exigen los videojuegos generalmente, quienes de por sí demandan un altísimo poder de procesamiento que sólo tales dispositivos proporcionan. Sin embargo, la comunidad científica junto con los fabricantes de estos chips colaboran para crear ambientes en los que no sólo los videojuegos aprovechen del poder de procesamiento de estos componentes, sino también otro tipo de tareas, tales como aquellas relacionadas con e-Science. Los clusters y grids tradicionales generalmente basan su procesamiento en las CPU (Central Processing Unit), sin embargo la comunidad científica no duda en utilizar estas arquitecturas distribuidas con procesamiento por GPU para optimizar la ejecución de algoritmos que demanden de HPC.

Varios investigadores del NCSA (National Center for Supercomputing Applications) de la Universidad de Illinois desplegaron y condujeron pruebas sobre dos potentes clusters basados en procesamiento por GPU. Los clusters son nombrados Lincoln y AC respectivamente, ambos compuestos de GPUs Nvidia Tesla S1070. AC consiste de 32 nodos HP, donde dichos nodos se componen de 2 procesadores AMD Opteron Dual-Core de 2.4 GHz y 8 GB de memoria principal, mientras que AC se conforma por 32 servidores Dell, cada uno con 2 procesadores Quad-Core Intel Xeon de 2.3 GHz de frecuencia y 16 GB de memoria principal [7].

Varias de sus aplicaciones fueron utilizadas para analizar el desempeño de los clusters GPU contra clusters CPU. Se utilizaron tres tareas e-Science para tales pruebas. Se empeezó con pruebas de una aplicación que estudia la distribución de los objetos en la esfera celeste, dicha tarea se ejecutó en el cluster predecesor de AC, QC, quien goza de una razón de 1 núcleo de CPU por GPU. Se utilizaron 48 GPUs vs 48 núcleos de CPU, y los resultados exponen que la ejecución en las GPU sobrepasó 30 veces las de las CPU. La segunda tarea utilizada como prueba fue un algoritmo que simuló sistemas biomoleculares y la tercera fue un algoritmo que elaboró cálculos energéticos. Ambas fueron ejecutadas en AC y Lincoln respectivamente, estas emplearon una razón 1:1 de núcleos CPU y GPUs en ambos clusters. Los resultados de las ejecuciones en ambos casos muestran los desempeños GPU como los mejores [7].

Existen situaciones en las que los interesados en correr tareas e-Science no cuentan con la toda la ciber infraestructura necesaria para llevar a cabo sus actividades, es por eso que empresas como Amazon y Pi-Cloud ofrecen servicios en la nube que permiten a los investigadores aprovechar los recursos de estas empresas. Chen, Liu, Ma y compañía, realizan un estudio en 2011 en el que exponen un análisis costo-beneficio sobre este tipo de servicios HPC en la nube versus el uso de clusters locales para la ejecución de algoritmos paralelizados.

Los autores del estudio emplean el servicio Amazon EC2 CCI, el cual brinda a sus usuarios flexibilidad para configurar las características de la ciber infraestructura donde se va a ejecutar su tarea. Alquilan nodos de 2 procesadores Intel Xeon Quad-

Core con 32 GB de memoria principal, interconectados con 10 GigaBit Ethernet. En el caso del cluster local, este se apoda como IB-cluster, y se componía del mismo número de nodos y similares especificaciones técnicas a las del cluster de Amazon. Para el análisis se conducen varias pruebas medidas a través de Amazon Linux AMI, el compilador Intel, e Intel MPI [8].

Se emplean tres aplicaciones de la vida real para las pruebas: la primera fue sobre un simulador de circulación oceánica, la segunda sobre un simulador de predicción climática y finalmente sobre un algoritmo de búsqueda secuencial de ADN. Los resultados de las pruebas a nivel de desempeño son muy parejas entre ambas configuraciones de ciber infraestructura, aunque los investigadores resaltan que cuando la nube necesita hacer uso de conjuntos de datos de cierto tamaño esta presenta algunas deficiencias de desempeño. Por otra parte, a nivel de costos se establece que la utilización de la ciber infraestructura de Amazon es totalmente viable ya que las estimaciones de costo por instalación y mantenimiento de un cluster local exceden a las del alquiler de la ciber infraestructura de Amazon, especialmente, si los usuarios de dichos clusters generalmente ejecutan tareas de baja carga de trabajo [8].

Como se aprecia en estas experiencias académicas, el aprovechamiento de la ciber infraestructura como clusters y grids a nivel hardware efectivamente aportan beneficios al momento de suplir computación de alto desempeño para tareas e-Science. Sin embargo es de vital importancia resaltar que no se va a obtener ejecuciones de tareas e-Science completamente exitosas sólo por el hecho de utilizar estas infraestructuras físicas y middleware como Globus o gLite. Por exitosas se aclara como el tiempo de ejecución de la tarea se vea reducido en el mayor grado posible, y esto sólo puede ser alcanzado si a los dos factores anteriormente mencionados se les añade el factor de programación paralela. Existen dos modelos de programación paralela tradicionales definidos, programación por hilos, cuando se utiliza ciber infraestructura de memoria principal compartida, y programación por paso de mensajes, cuando se utiliza ciber infraestructura de memoria principal distribuida.

Tales modelos de programación paralela se usan en dependencia de la ciber infraestructura hardware sobre la que se vaya a trabajar. Generalmente se emplean implementaciones MPI (Message Passing Interface) al momento de ejecutar algoritmos en sistemas distribuidos, tales como clusters y grids, mientras que las implementaciones con hilos/OpenMP son reservadas para algoritmos que se ejecutan en máquinas de memoria compartida, tales como supercomputadores, servidores, computadores de escritorio y portátiles. Existen también implementaciones para GPUs mediante librerías de Nvidia como CUDA o símiles de OpenMP como lo es OpenACC. Como dato adicional, también existen implementaciones híbridas (MPI/Hilos-OpenMP y CUDA/MPI) con la que es posible extraer la mayor cantidad de desempeño de una configuración de memoria principal distribuida [9].

Existen ejemplos a lo largo de la literatura donde grupos de investigadores aprovechan grids y clusters institucionales/comerciales para la ejecución de tareas e-Science mediante programación paralela. Borrell, Gorobets y compañía exponen sobre cómo la adaptación híbrida OpenMP/MPI de su solver 3D de Poisson logra superar el rendimiento de la versión MPI [10]. Cortez, Donate, González y compañía escriben sobre como optimizar los tiempos de respuesta de un algoritmo de predicción de series temporales con redes neuronales artificiales mediante una implementación paralela MPI [11].

Pope, Fitch, Pitman, Rice y Reumann exponen sobre la eficiencia de modelos de programación híbrida al momento de simular modelos fisiológicos multi-escala del corazón [12]. Jiang Murong, Chen Lin, Zhou Jun, Wang Dan y Tang Fang también describen sus experimentaciones con modelos de programación paralela híbrida OpenMP/MPI y OpenMP sólo, con el objetivo de llevar a cabo procesamiento de imágenes mediante ecuaciones diferenciales parciales [13]. Por otro lado, Jones y Yao proponen modelos de programación paralela híbrida OpenMP/MPI para mejorar la eficiencia de un algoritmo respecto a la generación de tomografías por OSEM (Ordered-Subset Expectation Maximization) [14].

Las experiencias mencionadas en los dos párrafos anteriores estaban relacionadas al aprovechamiento de los procesadores en clusters/grids, sin embargo cómo se

expresa anteriormente en esta sección, los científicos también optan por explotar otros recursos como los son las GPU. Los autores McConnell, Sturgeon, Henry, Mayne y Hurley evaluan el modelo de programación híbrido CUDA/MPI para un cluster GPUs en el cual optimizan la ejecución de un algoritmo de mapas de auto-organización [15]. Adicionalmente, Rengan Xu, Sunita Chandrasekaran y Barbara Chapman exponen múltiples casos de estudios en donde la implementación de un modelo híbrido OpenACC/OpenMPI para aprovechamiento de CPU y GPU en un nodo de un cluster [16].

Los investigadores recién mencionados exponen resultados y conclusiones de alto nivel en sus artículos de investigación, mas no profundizan de manera minuciosa los medios a través de los cuales logran sus objetivos, por ejemplo, no detallan o argumentan respecto a patrones, consideraciones o características a nivel de programación que permitan con mayor facilidad a otro colega intentar replicar u optimizar los experimentos, o ejecutar tareas de similar naturaleza; y es que la programación paralela como tal no es un concepto sencillo de abordar. ¿Por qué no dar pautas en los artículos sobre cómo concretamente practicar la programación paralela para la replicación de resultados o ejecución de tareas similares, a sabiendas que la programación paralela requiere de cierto nivel de estudio para explotarla, y sus usuarios generalmente no son versados en el tema?

En un artículo publicado en 2009 en ACM, titulado A View of the Parallel Computing Landscape, sus autores realizan un repaso sobre diferentes situaciones que enfrenta la comunidad científica a la hora de aprovechar la ciber infraestructura HPC distribuida para computación paralela. Entre las situaciones identificadas, los autores destacan que una de las claves para que cualquier investigador pueda aprovechar exitosamente la computación paralela, es que este pueda tener acceso a herramientas masificadas, que mediante capas de software le oculten detalles de implementación que probablemente desconozca y en consecuencia no sepa aprovechar, pero que en consecuencia le permitan modificar partes que pueda interpretar y explotar con comodidad. En otras palabras, no todos los investigadores que desean aprovechar ciber infraestructuras cluster/grid para HPC tienen la experticia para desarrollar algoritmos acordes al objetivo de maximización de desempeño de las tareas e-Science, por lo que es vital que los expertos en programación paralela alivien dichas carencias mediante la distribución de capas de

software de alto nivel, que los investigadores puedan manejar y aprovechar con mayor facilidad. Los autores del artículo describen como de la mano de patrones de diseño y expertos en computación paralela es posible encapsular software (el framework) para el adecuado aprovechamiento por parte de los investigadores/usuarios. Los frameworks de computación paralela exitosos agrupan soluciones a problemas recurrentes a través de experiencias colectivas de la comunidad programadora [17]. A continuación, se exploran algunos frameworks con respectivos conceptos asociados, y para finalizar se procede a concluir con la oportunidad de investigación que se puede extraer de esta colección de reportes y artículos.

En efecto el aprovechamiento de ciber infraestructuras como clusters y grids para tareas HPC se logra a través del adecuado uso/explotación de la programación paralela. Entre las mejores prácticas para llevar a cabo dicho aprovechamiento se encuentra el desarrollo de frameworks de programación paralela. Los frameworks de software básicamente buscan proporcionar librerías, generadores de código y asistencias que permitan abstraer al usuario final de complicados detalles de implementación [17].

El framework de programación paralela Horde es una contribución de ingenieros de sistemas y ciencias computacionales de la Universidad de Tsinghua en Pekín. Horde es un framework que busca facilitar el desarrollo de algoritmos paralelizados para HPC distribuida mediante el uso de grafos. Horde fue desarrollado para ofrecer fácil uso, tolerancia a fallos y portabilidad. La estructura de grafo se usa de manera que los vértices representan sub-tareas del algoritmo principal, emulan a MPI. Para que un usuario pueda aprovechar Horde éste primero debe programar los vértices del árbol, es decir las tareas que serán distribuidas en el cluster, luego debe elegir qué tipo de conectores se usarán, para tareas HPC los autores recomiendan conectores de memoria, luego el usuario debe ordenar la ejecución de las tareas mediante un planificador y finalmente ejecutar el grafo. Los autores de Horde llevan a cabo pruebas de desempeño del framework contra un algoritmo desarrollado en Boost Asio en donde los resultados arrojan a Horde como un aceptable competidor [18].

En un capítulo de la revista Scientific Programming de la IEEE, Joel Falcou hace un repaso sobre las virtudes que brinda la programación paralela hoy día pero adicionalmente resalta los errores en los que se incurre al implementarla de manera inadecuada. Falcou explica que en vista de la poca trivialidad que implica la programación paralela, ello conduce a los ingenieros de software y expertos en programación paralela a buscar alternativas que faciliten a otros la implementación de algoritmos para HPC distribuida. Entre estas alternativas, falcou hace mención sobre los esqueletos, unas piezas software que permiten implementación de algoritmos paralelos mediante la identificación de un patrón detectado en una gran cantidad de aplicaciones paralelizadas. Estos esqueletos le camuflan al usuario detalles de bajo nivel y este sólo tiene que preocuparse por escribir código secuencial. El esqueleto básicamente busca rebanar datos mediante un proceso maestro, mientras que procesos esclavos son los encargados de procesar dichos datos (analogía a MPI). Falcou destaca esqueletos como los de Kuchen, Murray Cole y el esqueleto Lithium, todos comprobados cómo buenas alternativas para programar algoritmos paralelizados de forma relativamente sencilla [19].

Para finalizar, Ritu Arora de la Universidad de Alabama publica en IEEE conceptos relacionados al aprovechamiento de la programación paralela. Describe cómo la abstracción, los patrones de diseño y DSL (Domain-Specific Languages) contribuyen a la creación de un framework bautizado FraSPa, en aras de proporcionar a usuarios finales medios eficientes a través de los cuales puede programar tareas para correr en clusters/grids. El framework busca resolver problemas que afrontan investigadores y otro tipo de usuarios de ciber infraestructura para HPC distribuido. Los problemas están relacionados con baja mantenibilidad, replicación, optimización y reingeniería intrusiva del código. La manera cómo el autor busca que el usuario aproveche el framework es de la siguiente manera: El usuario debe especificar la arquitectura para saber que DSL se va a necesitar para la ejecución de la tarea, a partir de esta especificación el framework selecciona las plantillas y código genérico correspondiente. El usuario sólo debe programar de manera secuencial de acuerdo a los lineamientos provistos por la instancia seleccionada de FraSPA, mientras que el framework se encarga de gestionar los detalles de bajo nivel para la ejecución de la tarea en un ambiente distribuido. Arora concluye al establecer criterios para determinar la eficiencia del framework y enuncia contribuciones de este tipo de software tanto para la ingeniería de software cómo para los propios usuarios del sistema [20].

Las bondades proporcionadas por las ciber infraestructuras software mencionadas, ya están comprobadas por los múltiples autores e inventores que las desarrollan o utilizan, pero es necesario dedicar una parte de esta sección del trabajo para destacar que dichas comprobaciones sólo son posibles mediante la utilización de modelos matemáticos y teorías científicas. Tales modelos y teorías se enmarcan dentro de las métricas de la ciencia del software, las cuales buscan valorar características del software HPC como lo son su desempeño y la complejidad de su implementación. A continuación se examina estado del arte respecto a teorías que estudian sobre la complejidad, facilidad, dificultad de desarrollo y mantenimiento de software desarrollado para HPC.

Si bien estas ciber infraestructuras software, tales como frameworks de programación paralela, y sus autores afirman sobre la facilidad de implementación de algoritmos paralelos a través de dichas herramientas, tales afirmaciones son o deben ser respaldada por teorías científicas y evidencia experimental. Maurice Halstead fue uno de los principales pioneros en el mundo de ciencias computacionales que propuso las primeras métricas respecto al desarrollo de software como esfuerzo, tamaño, volumen, dificultad, tiempo requerido, y las cuales hoy día se enmarcan dentro de la denominada Ciencia de Software [22]. Además de Halstead, Thomas McCabe, Shao y Wang contribuyen sus aportes en el campo con la instauración de la Complejidad Ciclomática [45], Índice de Mantenibilidad [43] y Tamaño Cognitivo Funcional respectivamente [23]. Existen otros modelos y métricas adicionales que permiten medir aspectos similares a los mencionados anteriormente, no obstante hay mayor aceptación en la comunidad científica para las métricas de Halstead ya que no se restringen por especificaciones a nivel lenguaje de programación, número de proyectos, líneas de códigos, entre otros [21].

El estudio de la complejidad del software en general es un tema que no encuentra verdades absolutas, ni acuerdos universales, pero se puede afirmar que entre las tres más recurrentes por diferentes autores y académicos son las métricas de Halstead, McCabe y el Índice de Mantenibilidad [23] [44], las cuales ayudan a determinar qué tan difícil o complejo es implementar y mantener un algoritmo.

A continuación, se procede a concluir sección de repaso del estado del arte, en referencia al aprovechamiento de ciber infraestructuras para desarrollar y ejecutar algoritmos HPC.

Con el análisis de las últimos párrafos de este capítulo sobre trabajos relacionados con la explotación de la programación paralela para ejecutar tareas e-Science en clusters y grids HPC, se identifica que los investigadores expositores generalmente no comparten estrategias, patrones o guías concretas a nivel de código, que permita a colegas u otros usuarios intentar replicar experiencias o trabajos similares a los descritos en los artículos de investigación, a sabiendas que la programación paralela es un paradigma de altísima complejidad para los no expertos.

Debido a esto y a los trabajos referidos en este capítulo respecto a aprovechamiento a nivel hardware y software de las ciber infraestructuras para e-Science, es adecuado desplegar un ecosistema completo de aprovechamiento para ejecutar tareas e-Science en clusters/grids HPC en la Universidad Tecnológica de Bolívar, y enfatizar en la creación de bases de un framework de programación paralela a partir de patrones específicos de software paralelo. Para iniciar, es necesario contar con ciber infraestructura para HPC, tales como clusters y grid, donde una aproximación oportunista puede brindar alto grado de aprovechamiento a nivel hardware, cómo en el caso UniAndes. Segundo, se debe precisar que la ciber infraestructura es de procesamiento por CPU ya que es una buena vía para comenzar a implementar este tipo de sistemas, sin embargo para próximos trabajos es deseable incluir procesamiento por GPU para sacar mayor provecho de los recursos con que cuenta la universidad. Tercero, una vez se tenga configurada la ciber infraestructura, lo siguiente es ejecutar las tareas e-Science en ella, sin embargo, existe conciencia de que la programación paralela es un asunto que se puede abordar, de manera que los usuarios del sistema la puedan aprovechar en alto grado, a pesar de que estos cuenten con limitaciones respecto al conocimiento y experiencia sobre el tema. Es por ello que se divisa como oportunidad de investigación la proposición de un framework de programación paralela interno en la universidad, que permita a los usuarios del sistema contar con alternativas y facilidades para desarrollar sus algoritmos de manera relativamente sencilla, sin que tengan que preocuparse por engorrosos y múltiples detalles de implementación.

Este tipo de ecosistema HPC de alto aprovechamiento no es implementado actualmente en la Universidad Tecnológica de Bolívar, lo cual brinda la oportunidad de no sólo ponerle en marcha, sino también de sentar las bases para la extensión de un ecosistema HPC más robusto del que se pretende desplegar inicialmente. El ecosistema en el medio a largo plazo puede comprenderse no sólo por clusters y grids locales, y bases de un framework de programación paralela para aprovechamiento de procesamiento por CPU, sino también de integración de servicios HPC en nube, cómo se vio en el caso de Amazon EC2, desarrollo de más frameworks de programación paralela para múltiples lenguajes, y para procesamiento por GPU, y adicionalmente, desarrollar interfaces web que abstraigan y facilite aún más el proceso de desarrollo de algoritmos e-Science de los investigadores/usuarios del sistema HPC de la UTB.

Cómo se aprecia en este capítulo del texto, existen ciber infraestructuras a nivel hardware y software que permiten el aprovechamiento de los recursos computacionales para optimizar la ejecución de tareas e-Science, y existen mecanismos que permiten determinar grado de aprovechamiento. Se presentan reportes y artículos investigativos en los cuales sus autores describen cómo mediante dichas ciber infraestructuras logran llevar a cabo sus procesos de estudio. Estos autores exponen sus datos y conclusiones, sin embargo no enfatizan sobre estrategias o patrones de software para el desarrollo de sus algoritmos, los cuales hacen uso del complejo paradigma de programación paralela, y dificultan la replicación de resultados, desarrollo y ejecución de algoritmos de similar naturaleza. Por ello, nos parece adecuado proponer la implementación de un ecosistema de aprovechamiento HPC en la Universidad Tecnológica de Bolívar, enfatizando en el desarrollo de bases de un framework de programación paralela, el cual le brinde a sus usuarios un ambiente alternativo de ejecución de tareas HPC de fácil acceso y manipulación.

A continuación, se presenta el objetivo general de este trabajo, seguido de los objetivos específicos del mismo, extraídos luego de analizar problemática y revisar el estado del arte respecto a dicha problemática.

2.3. OBJETIVOS

2.3.1. Objetivo General

Generar un ecosistema para aprovechar ciber infraestructuras al momento de desarrollar y ejecutar algoritmos e-Science que requieren de HPC en la UTB

2.3.2. Objetivos Específicos

1. Crear ambiente de desarrollo y ejecución a nivel hardware y software de algoritmos e-Science que requieren HPC en la UTB.
2. Producir bases de un framework a partir de patrones de programación paralela, de manera que algoritmos e-Science anteriormente codificados secuencialmente o mediante Hilos/MPI sean adaptables al mismo.
3. Evaluar a través de métricas software de McCabe, Halstead y Oman/Hagemeister, complejidad de programar algoritmo e-Science utilizando framework versus MPI puro en ecosistema de aprovechamiento.

2.4. HIPÓTESIS

El aprovechamiento de ciber infraestructuras HPC existentes en la UTB proporciona mayor facilidad y alternativas para el desarrollo y ejecución de tareas e-Science.

2.5. PREGUNTA DE INVESTIGACIÓN

¿Qué características de un ecosistema HPC permiten mejor aprovechamiento de ciber infraestructura hardware y software existente?

2.6. JUSTIFICACIÓN

Esta investigación se motiva por implicaciones prácticas, académicas y sociales. A nivel práctico se establece que los productos implícitos del ecosistema de aprovechamiento generarán mayor disponibilidad, alternativas y facilidades para el desarrollo y ejecución de tareas e-Science que requieren de HPC por parte de beneficiarios, en este caso, investigadores, estudiantes, profesores, y clientes externos de la UTB. A nivel académico, se piensa que lo anterior sienta bases para mayor profundización investigativa en materia de computación de alto desempeño a nivel de pregrado y posgrados en la UTB, esto visto en términos de profundización de investigación y aplicación de nuevas tecnologías HPC tales como APIs de programación, frameworks, networking, optimización de recursos computacionales y flujos de trabajo. Finalmente, a nivel social, avances en materia de aprovechamiento de ciber infraestructura HPC de la UTB, se reflejan en integración de usuarios externos, como compañías y estudiantes de otras instituciones académicas, lo que redunda en avance de desarrollo de productos o servicios que beneficien a la sociedad.

CAPÍTULO III: MARCO TEÓRICO

En esta sección del texto se exponen conceptos y teorías empleadas para diseñar la metodología de investigación de este trabajo.

3.1. E-SCIENCE

La e-Science se considera como prácticas científicas, (descubrimiento y organización sistemática del conocimiento), que requieren de almacenamiento y/o computación de datos. El término surge en 1999 gracias al entonces director general de la Oficina de Ciencia y Tecnología del Reino Unido, John Taylor, quien buscaba gestionar recursos para la ejecución exitosa de proyectos de esta naturaleza en dicha región. Los expertos en ciencias computacionales son aquellos que generalmente intermedian entre los cientificos y las tecnologías que estos necesitan en sus actividades e-Science. Existen e-Sciences como físicas de particulas, ciencias de la tierra y simulaciones sociales [55] [56].

El proyecto e-Science más ambicioso para la comunidad científica es el LHC (Large Hadron Collider) de CERN (European Organization for Nuclear Research). Se considera como una de las más grandes obras ingenieriles de la historia y opera en su total capacidad funcional desde 2009, reúne enormes cantidades de datos (con magnitudes del orden de petabytes) para ser almacenados y procesados en grids de más de 170 centros computacionales afiliados en más de 35 países de territorio europeo [55] [56].

Los ambientes y equipos de cómputo que permiten la práctica de e-Science se les conoce como ciber infraestructura. Estos brindan soluciones de conectividad de laboratorios, computadores, datos y sus derivados. Las ciber infraestructuras son aquellas tecnologías de la que los científicos manipulan y administran los datos de

sus procedimientos, y generalmente dichas infraestructuras se traducen en redes masivas como lo son los clusters y las grids [55] [56].

3.2. COMPUTACIÓN DE ALTO DESEMPEÑO (HPC: HIGH PERFORMANCE COMPUTING)

La HPC consiste en la agregación de poder computacional para resolver problemas de las ciencias, las ingenierías e industrias que no se pueden resolver con computación por PCs de escritorio o estaciones de trabajo típicas. La HPC se construye mediante el uso de supercomputadores tradicionales o redes escalables, tales como clusters y grids. Los supercomputadores tradicionales son equipos que cuentan con múltiples procesadores (CPUs), en ocasiones con más de un núcleo y alta cantidad de memoria principal. Otros supercomputadores no sólo procesan por CPU sino también por medio de potentes GPU que poseen muchísimos más núcleos que los procesadores más potentes y en consecuencia proveen mayor rendimiento que supercomputadores basados en CPU únicamente. Por otro lado, los clusters son la interconexión de varios PCs típicos que permiten ejecutar una tarea al usar recursos computacionales de todos ellos (memoria principal y secundaria, CPUs, GPUs). Las grids son la interconexión de varios clusters [57] [58].

Existe un concepto similar a HPC denominado HTC (High Throughput Computing). HTC se diferencia en HPC, en que este último procesa tareas que requieren de computación distribuida, es decir, una tarea se subdivide en pequeña tareas que son procesadas individualmente por un nodo de un cluster, y sus resultados son recolectados y exhibidos al usuario como una sola computación simple, mientras que HTC toma varias tareas individuales y las asigna a cada nodo de un cluster para procesarlas. Adicionalmente, HTC se utiliza para procesamientos que requieren de meses y años para su compleción, no así HPC, que por lo general no pasa de completar la tarea en unos cuantos días a lo sumo [57] [58].

3.3. ECOSISTEMA

El término ecosistema se asocia formalmente con la biología, [84] lo define como un entorno agrupado, de bajo acoplamiento, habitado por especies proactivas y reactivas respecto a beneficios propios, y que preserva su medio. Algunas instituciones de investigación y autores que enfatizan sus labores en el marco de las ciencias computacionales, emplean el término, de forma informal, para describir entornos computacionales en los que interactúa el hardware y software, para procesar datos con miras a brindar servicios [83]. El World Economic Forum (WEF) expresa una definición de alto nivel, como que los ecosistemas digitales son espacios formados por la convergencia de las industrias de medios, telecomunicaciones y tecnologías de la información, y se compone de usuarios, compañías, gobiernos, y sociedad civil, así también como la infraestructura que rige las interacciones digitales [86]. Por otro lado, los ecosistemas de tecnologías de la información (TI) de acuerdo a [85], son sistemas adaptativos complejos de sistemas autónomos, y resume que estos sistemas se adaptan a las necesidades de sus subsistemas y las necesidades de sus usuarios.

El término ecosistema se acompaña del término HPC, digital, TI, u otros, sin embargo estos agrupan términos recurrentes como especies (humanos, hardware), entornos (software), interacciones, necesidades y servicios. Un ecosistema computacional en el marco de este trabajo investigativo es aquél compuesto por ciber infraestructuras, servicios y sus usuarios.

3.4. CIBER INFRAESTRUCTURAS

Son sistemas de computación, almacenamiento de datos, instrumentos avanzados, entornos de visualización y personas, todas enlazadas a través de redes de alta velocidad, las cuales integradas apoyan la innovación y descubrimientos escolares. El término se relaciona a tecnologías de sistemas de información que proporcionan altas y avanzadas capacidades de procesamiento. El término se deriva de la palabra infraestructura, la cual las personas utilizan para “conectar y jugar” (plug & play), sin preocuparse por interconexiones y composición interna, concepto al cual se adhiere la ciber infraestructura, pero en este caso para facilitar el uso de servicios internamente robustos y complejos en su composición e interconexión [54].

3.4.1. Cluster

Los clusters son redes locales de computadores que proveen servicios como si fueran una sola máquina. Los cluster pueden ser de arquitecturas homogéneos o heterogéneos, es decir, si los computadores que lo componen tienen características similares o diferentes en cuanto a procesadores, GPUs, memoria principal y secundaria. Los clusters se componen de dos o más computadoras (o nodos) en donde uno de ellos se denomina 'maestro' mientras que los demás se llaman ´esclavos´; el nodo maestro es el que por medio de una capa de software específica gestiona el servicio ofrecido por el cluster a través de los esclavos [59] [60].

Los clusters tradicionalmente se emplean para ofrecer servicios como balanceo de carga y alta disponibilidad. El balanceo de carga busca que el cluster reparta tareas entre los nodos que lo componen, tales nodos ejecutan su tarea y entregan el resultado al maestro, el nodo maestro finalmente expone el resultado final ante el usuario que la solicita inicialmente. Por otro lado, la alta disponibilidad permite que en caso de falla de un esclavo al momento de procesar una tarea, otro nodo esclavo puede respaldarlo y ejecutar dicha tarea. Por supuesto, las tareas varían, desde solicitudes a web servers hasta computación de tareas HTC o HPC [59] [60].

Existen capas de software específicas que permiten el despliegue de servicios en cluster, paquetes como HTCondor, Torque, Microsoft Cluster Service, Apple QMaster, Globus Toolkit, Xgrid, 3Ds Max Backburner, Maya Satellite, sólo por nombrar, permiten la distribución de tareas a lo largo y ancho de una red local de computadores. Estos paquetes emplean interfaces gráficas o de líneas de comando para que sus usuarios puedan enviar sus tareas [59] [60].

3.4.2. Grid

Las grids son la interconexión de dos o más clusters para proveer aún más capacidad de computación al momento de resolver tareas. Las grids pasan de ser redes locales (LAN) a ser WANs. Las grids más ambiciosas a nivel mundial operan a través de internet, lo que brinda posibilidades de escalamiento de altas

magnitudes. Las grids operan similar a los clusters, sólo que la tarea que vaya a procesar la grid se subdivide en todos sus clusters asociados. Las grillas o grids para HPC funcionan como plataformas investigativas cuyos impulsores son los investigadores europeos de CERN, que a su vez crean las primeras herramientas de software que posibilitaban su operación, p. ej. Globus Toolkit y gLite [61].

Existen despliegues de grids HPC tipo colaborativa/oportunista (procesamiento en segundo plano), y dedicada (procesamiento prioritario) patrocinadas por entidades como UCB, CERN, quienes en su búsqueda de promover herramientas de libre acceso y de gran capacidad para brindar soluciones computacionales altamente distribuidas [61].

3.5. ARQUITECTURA DE SISTEMA SOFTWARE

Es la organización de un sistema que comprende elementos software, propiedades visibles de esos elementos y la relación entre ellos. El término propiedades visibles implica suposiciones que otros elementos pueden hacer de otro elemento, tales como servicios, características de desempeño, manejo de fallos, utilización de recursos compartidos, entre otros [82].

3.5.1. Requerimientos Funcionales

Son las funciones o servicios que un sistema software proporciona a uno o varios actores que intervienen en este. Los requerimientos funcionales son descritos en términos de comportamientos, los cuales a su vez se explican por entradas, secuencia de acciones y respectivas salidas [82].

3.5.2. Atributos de Calidad

Llamados también requerimientos no funcionales, estos definen como el sistema software debe operar. Estos atributos deben garantizar un comportamiento

constante del sistema bajo ciertas circunstancias, y permiten calificar un sistema software como bueno o malo [82].

3.5.3. Vistas

Son representaciones parciales del sistema con base en puntos de vistas. Dichos puntos de vistas son un conjunto de restricciones. Entre los las vistas que más se emplean para describir arquitecturas de sistemas se encuentran: Vista lógica, de desarrollo, física, procesos y escenarios, cada una de estas asociadas con diferentes diagramas y modelos [82].

3.5.4. Modelo de Negocio

Se enfoca el comportamiento y flujo de información dentro de la organización o sistema. Captura los eventos, entradas, salidas más significativas, asociadas a procesos de negocio relevantes. El modelo de negocio no hace parte de la arquitectura pero se utiliza para la validación del sistema, y para que el validador entienda el problema. Sirven de apoyo para entender la arquitectura [82].

3.5.5. Vista Lógica

Se enfoca en lo que el sistema debe proveer en término de servicios/funcionalidades a sus usuarios. Expresa la forma como el sistema atiende los requerimientos funcionales [82].

3.5.6. Vista de Procesos

Se enfoca en aspectos no funcionales de la arquitectura en relación a la vista lógica. Representa comportamiento del sistema a través del tiempo, resaltando aspectos de concurrencia, secuencialidad. Se representa con diagramas de tiempos, secuencia, estado, colaboración, comunicación [82].

3.5.7. Vista de implementación

Se enfoca en la organización del módulo de software del sistema. Representa forma cómo el producto software es implementado y ello se describe a través de diagramas de paquetes [82].

3.5.8. Vista Física

Se enfoca en aspectos no funcionales de la arquitectura. Representa como el sistema se despliega en componentes físicos para poder cumplir funcionalidad para el cual fue construido, resaltando aspectos de topología y comunicaciones, se representa con diagrama de despliegue [82].

3.5.9. Vista de Escenarios

Representan casos de uso y escenarios de atributos de calidad de la arquitectura del sistema software [82].

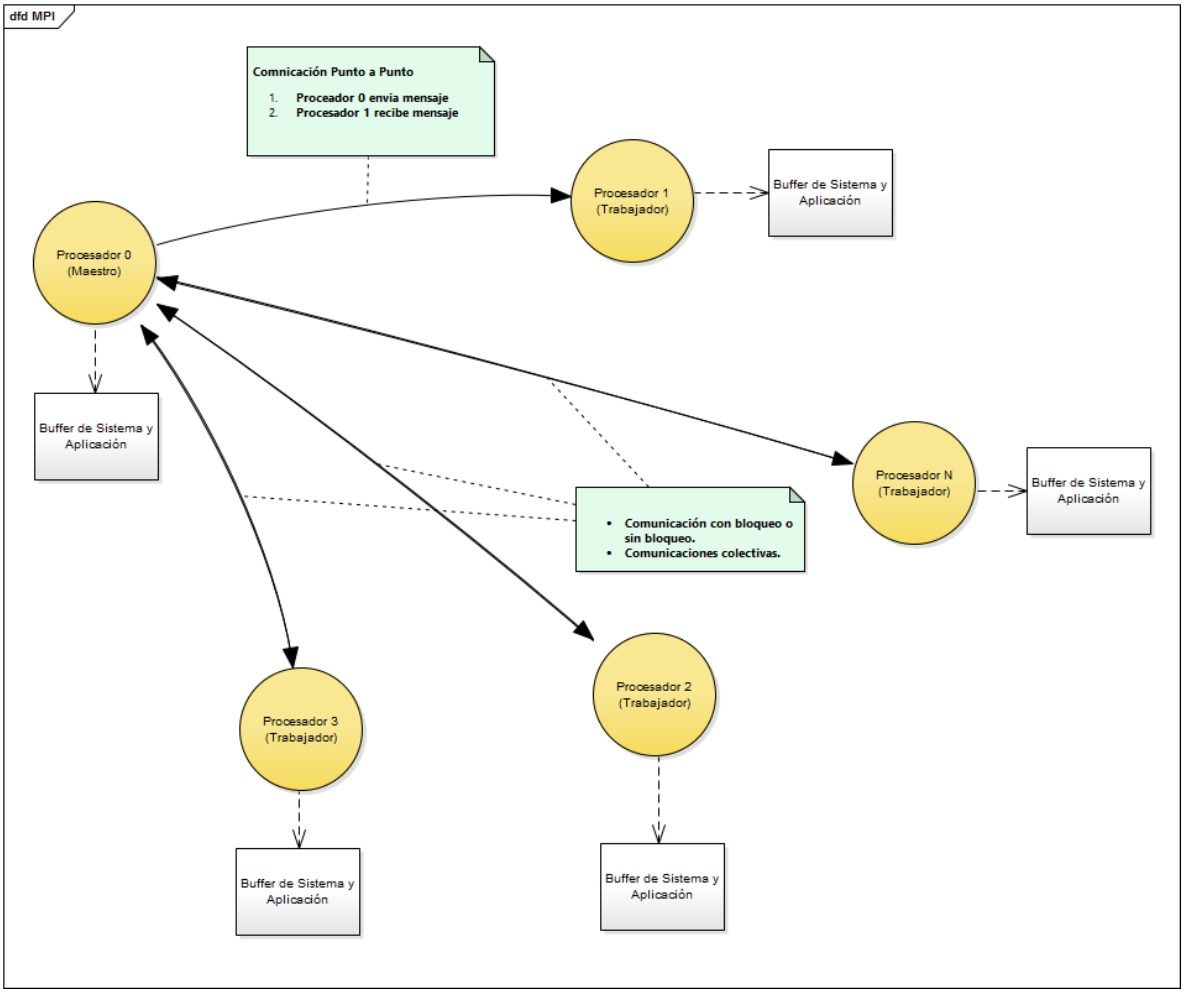
3.6. PROGRAMACIÓN PARALELA

Es una forma de programación en la cual se hace uso de arquitecturas por memoria compartida y paso de mensajes para ejecutar varias subtareas de una tarea en paralelo. La programación paralela es uno medio utilizado por la HPC para alcanzar su fin ulterior de optimizar el tiempo de ejecución de una tarea. El esquema bajo el cual funciona este paradigma es el de “Divide y Vencerás”, hace alusión a que una tarea se divide en varias subtareas que se pueden ejecutar paralelamente. Este paradigma es soportado por lenguajes de programación como C, C++, Java, Python, y otros. Las tecnologías básicas que permiten la implementación de este modelo de programación son los hilos y el paso de mensajes [62].

3.6.1. MPI

La tecnología MPI (Message Passing Interface) consiste en un conjunto de interfaces que permiten la implementación de programación paralela en entornos distribuidos. Por entornos distribuidos se entiende clusters o grids (procesadores o GPUs distribuidas), en las cuales las subtareas de una gran tarea se ejecutan paralelamente en los diferentes computadores del cluster o grid. MPI se utiliza para ejecutar programas paralelos desarrollados para aprovechar tanto CPUs como GPUs distribuidas. Las implementaciones MPI más utilizadas son MPICH y OpenMPI [63].

Figura 1: Esquema de Funcionamiento de MPI



MPI funciona designando un nodo/procesador como el ´Maestro´, el cual gestiona el control del algoritmo. El nodo Maestro puede enviar y recibir mensajes (datos) a otros nodos ´Trabajadores´ para que estos procesen código fuente y datos, dependiendo de la necesidad del usuario. Los mensajes pueden ser comunicados punto a punto o colectivamente. Durante el paso de mensaje se puede controlar la sincronización y 'hambruna de recursos' de los nodos, mediante operación de bloqueo o no-bloqueo; esto puede controlar la finalización de una operación de comunicación punto a punto mediante el uso del buffer de sistema por parte de los nodos. El buffer de sistema almacena datos que esperan ser procesados por el buffer de aplicación de los nodos. La Figura 1 exhibe un esquema del funcionamiento de MPI [48].

3.6.2. Descomposición

Se conoce como la identificación de paralización de un problema. Una tarea puede descomponerse en subtareas y ser procesados concurrentemente. Se debe determinar si la solución paralela se puede conseguir con más rapidez que la secuencial. Este se conoce como la primera instancia de la paralelización. No todos los problemas son paralelizables. Por ejemplo, el cálculo de la serie de Fibonacci no es paralelizable por la alta dependencia que existe en las operaciones implicadas. Existe descomposición de dominio, en la cual se busca dividir un conjunto de datos en tareas individuales, y descomposición funcional, en donde se enfoca en la división de operaciones [32] [74].

3.6.3. Distribución

La explotación física de la potencial paralelización identificado por la descomposición, resulta en el mapeo estático o dinámico de operaciones de tareas en procesadores [32].

3.6.4. Reparto de Código y Datos

Durante la descomposición y distribución de operaciones en los procesadores, datos y código son procesados de forma independiente o dependiente. Es común que existan procesadores que comparten el mismo recurso computacional [32] [75].

3.6.5. Comunicación

Se refiere a protocolos o mecanismos que posibilitan la interacción entre dos procesadores en el marco de redes computacionales. La comunicación es un factor preponderante dentro de la computación de alto desempeño, ya que características

como topología, número de nodos, distancias, sincronización, ancho de banda, latencia y otros, condicionan el desempeño de una red computacional [75] [76].

3.6.6. Sincronización

Es la secuencia de ejecución de operaciones que puede incidir en el desempeño del procesamiento de una tarea a través de una red computacional. La sincronización en el marco de la HPC, se divide en operaciones sincronizadas, es decir, una emisor que realiza una comunicación debe esperar que su receptor envie al emisor la respuesta, mientras que en las operaciones asíncronas el emisor puede seguir comunicándose con otros receptores sin necesidad de esperar detenerse a esperar la respuesta de estos [32] [77].

3.7. ABSTRACCIÓN

Es una técnica utilizada en ciencias computacionales para representar elementos de alta complejidad o especificidad en términos de otros de menor complejidad o más generales. Este concepto se utiliza, entre otros, en diseño y programación de software [73].

3.7.1. Software Framework

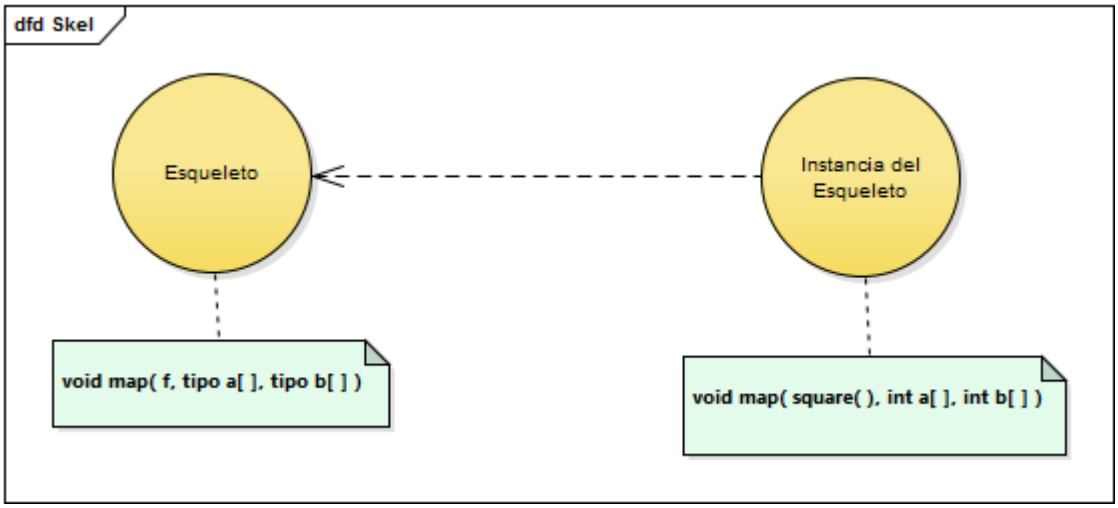
Los framework son un conjunto de librerías que proveen un servicio predeterminado pero que puede ser altamente extensible por su usario de acuerdo a sus necesidades. Los frameworks le evitan que los desarrolladores tengan que incurrir en implementaciones de bajo nivel para que puedan concentrarse directamente en los requerimientos que necesitan programar. Estas herramientas son utilizadas en numerosos campos de aplicación dentro del mundo del desarrollo de software, desde la industria del diseño gráfico, desarrollo web, aplicaciones empresariales hasta aplicaciones HPC, multimedia, middlewares, entre otros [71].

3.5.1.1 Esqueletos Algorítmicos

Los esqueletos algorítmicos son librerías o frameworks de programación paralela que se encargan de esconder detalles de implementaciones MPI-Hilos/OpeneMP a sus usuarios, de forma que estos puedan programar con mayor facilidad sus aplicaciones paralelas. Entre las ventajas de estos tipos de frameworks se encuentra: menos ocurrencia de errores por manipulaciones de bajo nivel y que la sincronización y orquestación de actividades paralelas se da a través de los patrones del esqueleto. Los esqueletos se encargan internamente de controlar aspectos de programación paralela, tales como descomposición, distribución, comunicación, reparto de código y datos, y sincronización [32] [72].

Los esqueletos algorítmicos en general se soportan bajo el concepto de las funciones de orden superior. Estas funciones se destacan por recibir como parámetro de entrada, por lo menos, una función, junto a otros parámetros, ya sea tipo datos u otras funciones, y adicionalmente regresa como salida otra función. Un ejemplo de este tipo de función es la función ‘mapeo’, la cual recibe como entrada una función cualquiera, un conjunto de datos, y como salida devuelve el mismo conjunto de datos, pero cada elemento del conjunto sale transformado por la función de entrada que recibió la función mapeo. Por ejemplo, la función de orden superior ‘MAP(SQUARE, int a[ ], int b[ ])’ recibe como entrada la función cuadrado, una lista de números enteros, y regresa como salida la función cuadrado aplicada al conjunto de datos a, representado esto en la lista b [32]. En la Figura 2 se aprecia esquema de funcionamiento del concepto de esqueleto algorítmico basado en funciones de orden superior.

Figura 2: Esquema de funcionamiento de un Esqueleto Algorítmico basaodo en Funciones de Orden Superior



3.5.1.1.1 Esqueleto Fixed Degree Divide & Conquer

El esqueleto sigue los principios de la estrategia Divide y Vencerás, la cual establece que un problema puede subdividirse recursivamente en instancias más pequeñas hasta que el problema sea indivisible, con lo cual se aplica un función base sobre todas las instancias del problema. Esta estrategia permite la implementación de soluciones concurrentes para las diferentes instancias de un problema. El esqueleto algorítmico en este caso mapea las diferentes sub instancias del problema en procesadores, jerarquizados en un árbol de procesos, donde estos son procesados concurrentemente [32].

El esqueleto FDDC se soporta sobre una secuencia de operaciones tales como 'indivisible', la cual verifica hasta qué punto un problema es divisible, 'f', que es la función personalizada que el usuario final aplica al caso base del problema, 'split', método que divide el problema en sus instancias y finalmente 'join', la cual une las soluciones de las sub instancias del problema. FDDC existe en varias versiones, una de estas es la versión binaria. La versión binaria de FDDC establece que las operaciones mencionadas anteriormente, en su mayoría, dependen de un parámetro k, con valor 2 (k=2). Esta constante se utiliza para mapear los procesadores involucrados en un árbol binario completo o árbol H, el cual hace uso de fórmulas algebraicas para determinar que procesadores enlazan con cuales. Las demás variantes de FDDC varían en el valor de la constante k, por ejemplo, 4 y un

valor genérico, y en consecuencia las formas de mapear los procesadores en un árbol, que esta vez no es binario [32].

3.8. MÉTRICAS DE SOFTWARE

Las métricas se emplean para cuantificar características específicas de un sistema software. Se logra medir propiedades como número de líneas de código, grado de acoplamiento cohesión, complejidad ciclomática, índice de mantenibilidad, métricas Halstead, tiempo de ejecución, y otros. Tales mediciones se establecen en aras de incorporar la ciencia en la rama de la computación. Gran cantidad de estas métricas no logran ser completamente estandarizadas por la naturaleza compleja y variada de los sistemas software [78].

3.8.1. Complejidad de Programación

Cubre área del estudio del software relacionada a la creciente dificultad de entendimiento y predicción de aspectos de un sistema software a medida que este crece. Muchos autores proponen teorías específicas, Halstead, McCabe, Henry, Kafura, Chidamber, Kemerer, entre otros [79].

3.8.1.1. Métricas Halstead

Maurice Halstead propone en 1977 una serie de métricas con el propósito de exponer el software como parte de la ciencia. Propone múltiples métricas interrelacionadas que evalúan una pieza de software independiente del lenguaje de programación utilizado. Entre ls métricas se destaca número de operadores y operandos, volumen, tiempo aproximado de programación, esfuerzo de programación, dificultad de entendimiento de software, y otros. Muchos autores debaten sobre la precisión de dichas métricas y la de otros autores, pero muchos coinciden en que Halstead propone algunas muy válidas [80].

3.8.1.2. Complejidad Ciclomática

Las métricas de McCabe, agrupadas bajo la Complejidad Ciclomática (CC), matemáticamente establecida como V(g), son otro conjunto de medidas que determinan el grado de complejidad de piezas software con base en decisiones explícitas en el código fuente del software [81].

3.8.1.3. Índice de Mantenibilidad

Oman y Hagemeister proponen la métrica de complejidad conocida como el Índice de Mantenibilidad (MI), el cual mide que tan fácil se presupone es mantener una pieza software. Esta métrica es avalada por sectores académicos y de la industria de tecnologías de la información [43] [50].

A continuación, en el siguiente capítulo de este escrito se especifican criterios metodológicos tenidos en cuenta para ejecutar los objetivos específicos de este trabajo, y que conllevan a la consecución del objetivo general. Cada objetivo específico se compone de actividades que deben ser ejecutadas secuencialmente, a excepción de actividades del segundo objetivo específico que se pueden ejecutar de manera paralela e iterativamente.

CAPÍTULO IV: METODOLOGÍA

Esta fase del trabajo investigativo busca unas recolecciones de datos con base en las salidas producidas por la ejecución del segundo y primeras dos actividades del tercer objetivo específico; se hace referencia a objetivos de: producción de un framework de programación paralela inspirado en concepto de esqueletos algorítmicos y evaluación a través de métricas software de la complejidad de programar algoritmo e-Science utilizando framework versus MPI puro. Igualmente, el segundo objetivo específico es dependiente de la correcta ejecución del que le precede, despliegue de cluster MPI. A continuación, se detalla por objetivo específico las actividades requeridas para la compleción de cada uno de los mismos. Se argumenta sobre criterios de selección de cada actividad, ello incluye, elementos que intervienen, tales como actores, entradas, operaciones, dependencias, medios y salidas.

4.1. CREAR AMBIENTE DE DESARROLLO Y EJECUCIÓN A NIVEL HARDWARE Y SOFTWARE DE ALGORITMOS E-SCIENCE QUE REQUIEREN HPC EN LA UTB

La ciber infraestructura HPC existente de la Universidad Tecnológica de Bolívar no proporciona suficiente disponibilidad para que usuarios finales como investigadores, estudiantes, profesores, efectúen tareas e-Science de forma concurrente, lo que deriva en necesidad de analizar, diseñar, implementar, probar y desplegar tecnologías hardware y software que posibilite crear y ejecutar estas tareas e-Science en ciber infraestructura de la universidad con mayor provecho de lo que se cuenta. Se establece cuatro actividades que posibilitan lo anterior, las cuales se detallan a continuación y son ejecutadas por diseñador/implementador/administrador del ecosistema de aprovechamiento.

4.1.1. Diseñar y Describir arquitectura del sistema

Los sistemas de software modernos se caracterizan por contar con documentación formal relacionada a su concepción, cumplimiento de requerimientos funcionales y atributos de calidad, estrategia de implementación y despliegue, entre otros. A dicha documentación se le denomina Documento de Arquitectura de Software, que consecuentemente contiene toda la información pertinente a la arquitectura software de un sistema. La arquitectura de un sistema software registra decisiones arquitectónicamente significativas que expone cómo se satisface atributos de calidad, como lo son desempeño, disponibilidad, seguridad, mantenibilidad, portabilidad y usabilidad, así también como los requerimientos funcionales [24].

En el contexto del ecosistema de aprovechamiento HPC que se propone, se establece que corresponde diseñar y describir/documentar la arquitectura del mismo, justificado en necesidad de contar con material arquitectónico que permita a actores que intervienen en sistema tener claridad de sus roles, contar con información precisa de múltiples aspectos del sistema, tal como alcance, detalles de implementación, alcance, entre otros, y adicionalmente, a corto, mediano y largo plazo se pueda hacer revisiones, modificaciones, extensiones del sistema de manera fácil y formal [24].

Los diseñadores, implementadores y futuros administradores del sistema son los encargados de construir tal texto. A continuación se detalla actividades requeridas para diseñar y describir arquitectura del sistema acorde a directrices de Software Engineering Institute [25]:

- La arquitectura software del sistema requiere como primer paso identificar los tipos de actores o interesados que hacen uso del sistema de acuerdo a diversos intereses.
- Estructurar vista general de sistema de acuerdo a necesidades de los actores y contextos asociados.
- Identificar requerimientos funcionales del sistema y sus atributos de calidad.
- Establecer escenarios generales y críticos de atributos de calidad.
- Identificar aproximaciones arquitectónicas, tales como tipo de aplicación, estilos y patrones arquitectónicos, aspectos transversales, tecnologías.

- Documentar vistas arquitectónicas, tales como, vista de negocio, lógica, de procesos, de despliegue.
- Establecer relación entre vistas arquitectónicas.
- Realizar validación de la arquitectura, eso incluye, establecer árbol utilitario, identificar riesgos, no-riesgos, puntos de compensación. La validación se efectúa a través de metodología ATAM.

En el punto del diseño y descripción de la arquitectura en el que se debe especificar tecnologías u herramientas concretas que materialicen la arquitectura, se determina elegir aquellas que sean de libre distribución y alta documentación, lo que permite fácil reproducción, modificación y escalación de todas las actividades realizadas en el marco de este trabajo investigativo.

4.1.2. Configurar cluster MPI

La configuración de cluster MPI se aborda una vez existe claridad y consistencia en el documento de arquitectura del sistema. Al determinar tecnologías, sus interconexiones, dependencias y procedimientos de implementación, es posible abordar implementación de la ciber infraestructura cluster MPI. El cluster permite que usuarios finales ejecuten el código de sus algoritmos paralelos de forma distribuida, es decir repartir segmentos de código en diversos computadores que los procesan paralelamente, en general se reduce el tiempo de ejecución de dichos algoritmos versus ejecutar su código de forma secuencial, pero esto depende en gran medida al diseño e implementación del algoritmo [26].

El documento de arquitectura del sistema exhibe especificaciones hardware respecto al despliegue final de los sistemas. Por restricciones de logística y en búsqueda de agilidad en proceso de implementación de ciber infraestructura, parte de las tecnologías se despliegan en infraestructura hardware de entorno de pruebas/experimentación y otra en entorno de despliegue. Posterior a pruebas/experimentos, la ciber infraestructura se adapta a entorno de despliegue, es decir, en instalaciones de la UTB. Sólo se requiere adaptar ciber infraestructura a nuevo hardware ya que el alto grado de portabilidad del sistema facilita tal

procedimiento. En el despliegue final este cluster consta físicamente de servidores y/ computadores y máquinas virtuales Linux alojadas en los mismos.

Para la configuración de cluster MPI se requiere de las siguientes actividades [26] [27] [3], las cuales lleva a cabo el diseñador/implementador/administrador del sistema:

- Identificar y procurar procesadores/computadores que componen el cluster
- Crear máquinas virtuales acorde con especificaciones hardware indicadas en documento de arquitectura de software. Las máquinas virtuales
- Instalar distribución Linux en máquinas virtuales. Configurar usuario común en cada máquina virtual. El usuario en común se configura para efectos de facilitar configuraciones de almacenamiento en red y acceso remoto a otros computadores del cluster.
- Configurar red local con los computadores/procesadores del cluster. Una de las características más sobresalientes de cualquier tipo de cluster es la capacidad de comunicación, la cual es lograda a través de tecnologías de redes locales.
- Instalar y configurar sistema de almacenamiento compartido NFS. Este sistema centraliza el almacenamiento de archivos en una ubicación compartida por todos los computadores del cluster, ideal para organizar archivos en una red local.
- Instalar y configurar acceso remoto a nodos del cluster por SSH. Este protocolo de comunicación permite trabajar con fluidez en sistemas distribuidos, y es necesario para el despliegue de tecnología de comunicación por paso de mensajes.
- Instalación y configuración de interfaz de paso de mensaje MPI. Es la tecnología central de este proyecto con la cual se evalúa algoritmos e-Science.

4.1.3. Configurar middleware para integrar cluster y grid.

Un middleware integrador de servicios de computación de alto desempeño HPC/HTC permite unir varias tecnologías asociadas bajo una interfaz común, esto

implica manejo de contados conceptos y herramientas para el aprovechamiento de diversos servicios heterogéneos en implementación y sintaxis, pero homogéneos en naturaleza [27]. Se determina que la instalación y configuración de middleware de computación de alto desempeño HTCondor es una opción valiosa en el contexto de la integración de servicios cluster y grid a nivel e-Science en Colombia, ya que la red académica nacional 3COA de Renata se apoya sobre HTCondor y otras tecnologías, y la Universidad Tecnológica de Bolívar hace parte de la red de Renata. Condor como popularmente se le conoce emplea diversos paradigmas HPC como programación paralela por memoria distribuida, por memoria compartida, por lotes, servicios grid, y todos ellos en el marco de varios lenguajes de programación como Java, C, C++, Python, entre otros [30].

4.1.4. Configurar herramienta de monitoreo de red.

La Instalación y configuración de monitor de red local es importante. Un monitor de red local como Munin por ejemplo posibilita valorar estado de transferencia de datos en un cluster, throughput utilizado, nivel de procesamiento, espacio en disco, y es un recolector de datos que a la postre sirve como herramienta para análisis estadístico respecto a uso de recursos computacionales del cluster. Este tipo de herramientas también facilita la identificación y resolución de problemas en la red. Munin es una herramienta libre y muy utilizada en la comunidad Red Hat, distribución base de la cual se desprende la distribución Linux CentOS [27] [3] [30].

4.2. PRODUCIR BASES DE UN FRAMEWORK A PARTIR DE PATRONES DE PROGRAMACIÓN PARALELA, DE MANERA QUE ALGORITMOS E-SCIENCE ANTERIORMENTE CODIFICADOS SECUENCIALMENTE O MEDIANTE HILOS/MPI SEAN ADAPTABLES AL MISMO

Varios autores exponen sobre la dificultad que implica para aprendices o inexpertos dominar la teoría de paso de mensajes o programación paralela en entornos distribuidos. En respuesta a ello estos y otros autores proponen soluciones software que buscan minimizar la dificultad o complejidad anteriormente mencionada [31] [32]. En el mundo del desarrollo de software existen decisiones de diseño que al implementarse en código suben el nivel de abstracción de dicho código y ocultan

detalles de implementación específicos con el objetivo de que estos sean invisibles para otro desarrollador que programa sobre este código [31] [32] [33] [34].

De la palabra ecosistema, en su definición acorde a la RAE [36], se desprenden términos como ‘comunidad de seres’ y sus ‘relaciones en un mismo ambiente’. Se hace analogía de esta definición respecto al ambiente computacional y los “seres que en este conviven”, como lo son en este contexto piezas hardware y software. En el primer objetivo específico de este proyecto se expone sobre metodología referente a instalación y puesta en marcha de ciber infraestructura hardware HPC (cluster MPI) que permita a usuarios finales ejecutar algoritmos paralelos, por lo tanto se propone complementar el ecosistema con una pieza software para HPC con base a características que permita a usuarios finales desarrollar algoritmos paralelos con mayor facilidad/menor dificultad.

Se desea producir pieza software sobre la cual investigadores, estudiantes, profesores de la UTB escriban sus algoritmos e-Science y tenga la opción de omitir estudio e implementación de teoría y librerías de programación MPI y así sólo enfocar en la lógica del problema que se busca solucionar. El diseñador/implementador/administrador del ecosistema de aprovechamiento realiza las actividades descritas a continuación con miras a producir dicha pieza de software.

4.2.1. Identificar decisiones de diseño a través de revisión de la literatura respecto a los esqueletos algorítmicos

En el mundo de la programación paralela existen estructuras o patrones software de alto nivel denominadas Esqueletos Algorítmicos, entre otros, que proporcionan herramientas para desarrollar algoritmos paralelos con menor dificultad que de la manera tradicional [31] [32], es decir, programar directamente empleando librerías de bajo nivel como MPI, Hilos, y otros. Se realiza revisión bibliográfica referente al tema ante la novedad que este implica a nivel conceptual para implementador del ecosistema. La revisión determina de forma sustancial aproximaciones de diseño respecto a la pieza software que se desarrolla como salida este segundo objetivo específico [31] [32] [33] [34].

4.2.2. Identificar decisiones de diseño a través de revisión de conceptos sobre MPI

Todo algoritmo que se ejecuta en un cluster de memoria distribuida y que procesa a través de sus núcleos, debe hacer uso de la interfaz de paso de mensaje MPI [35]. Esta interfaz expone un gran número de métodos y atributos para que los programadores desarrollen sus algoritmos e-Science y luego los ejecutan en clusters. Al igual que en la actividad que precede, los detalles de implementación, conceptos, teorías detrás de MPI son abordadas por el implementador/diseñador/administrador del ecosistema de aprovechamiento ante la novedad conceptual que estos conllevan. Así mismo, la actividad anterior brinda herramientas conceptuales para desarrollar la pieza software que produce como salida este objetivo específico. También es preponderante realizar revisión de concepto de esqueletos algorítmicos y la tecnología que los concreta a nivel software, MPI, y así elegir partes de la interfaz que mejor interpretan aproximaciones o decisiones de actividad anterior..

4.2.3. Programar framework en lenguaje C/C++ con tecnología MPI

Al identificar aspectos importantes de las revisiones sobre esqueletos algorítmicos y MPI, se prosigue a traducir esto en código fuente. Se identifica en primera actividad de este objetivo específico decisiones y secuencia de operaciones necesarias para obtener un esqueleto determinado. Posteriormente, se examina en API MPI los métodos y secuencias que permiten plasmar el esqueleto o símil en código fuente.

MPI es implementado en múltiples lenguajes, pero mayoritariamente difundido en lenguaje C, el cual es ampliamente conocido por académicos y desarrolladores [37]. Se determina escribir la pieza software o framework en lenguaje C++ para aprovechar ventajas de sintaxis y funcionalidad, como la programación orientada a objetos, en combinación con sentencias MPI en lenguaje C. Ambos lenguajes son perfectamente combinables ya que son parte de la familia de compiladores GCC, pero se requiere que ambos compiladores provenga del mismo distribuidor [38].

Por framework se entiende código que abstrae detalles de implementación y expone una interfaz más sencilla de usar, con un comportamiento predeterminado y altamente extensible o modificable [39]. Los esqueletos algorítmicos siguen esta premisa, por lo tanto se propone diseñar e implementar bases de un framework de programación paralela que utiliza como principio de diseño patrones de comunicación y distribución de código implícitos de los esqueletos algorítmicos. Se decide implementar inicialmente el framework con base en un esqueleto algorítmico por motivo de alcance en la ejecución del proyecto de Ecosistema de Aprovechamiento HPC, pero con la intención de que el framework suministre bases para desarrollar variados tipos de problemas e-Science.

4.3. EVALUAR A TRAVÉS DE MÉTRICAS SOFTWARE DE MCCABE, HALSTEAD Y OMAN/HAGEMEISTER, COMPLEJIDAD DE PROGRAMAR ALGORITMO E-SCIENCE UTILIZANDO FRAMEWORK VERSUS MPI PURO EN ECOSISTEMA DE APROVECHAMIENTO

Si bien autores afirman respecto a que piezas software de mayor nivel de abstracción facilitan el desarrollo de otras utilizando estas como base (reduce complejidad de programación), como por ejemplo los esqueletos algorítmicos [32] [41], es pertinente evaluar si para el caso del framework de paralelización que se desarrolla en el marco del Ecosistema de Aprovechamiento HPC de la UTB, aplica la premisa de que desarrollar un algoritmo, e-Science por ejemplo, mediante framework inspirado en esqueletos algorítmicos implica menor tiempo, esfuerzo, complejidad para el programador [32], o mejor aprovechamiento del ciber infraestructura software al momento de desarrollar sus algoritmos. Para realizar estas valoraciones, se llevan a cabo pruebas en entornos de ejecución de iguales condiciones de hardware, con miras a minimizar análisis y conclusiones sesgadas, a diferencia a como propone Cole, quien realiza sus pruebas empleando números desiguales de procesadores [32]. Autores de framework basado en esqueletos algorítmicos, Skelgis, realizan pruebas sobre su framework con base en métricas Halstead [31] [41].El implementador/diseñador/administrador del Ecosistema es quien realiza las respectivas valoraciones del código fuente; a continuación se resaltan las actividades que direccionan hacia la evaluación sobre el framework.

4.3.1. Programar y probar algoritmo e-Science en MPI puro y ejecutar en ciber infraestructura cluster MPI

En primera instancia se escribe un algoritmo e-Science mezclado con sentencias MPI. Se busca expresar el algoritmo de forma procedimental sin muchas estructuras adicionales que dificulten entendimiento del código. Una vez programa es escrito, se debe probar y ejecutar satisfactoriamente en cluster MPI.

4.3.2. Programar y probar mismo algoritmo e-Science en framework y ejecutar en ciber infraestructura cluster MPI

En segunda instancia se requiere programar el mismo algoritmo e-Science de la actividad anterior, pero esta vez utilizando framework de paralelización basado en esqueletos algorítmicos. Algunas partes estructurales del algoritmo están restringidas por la plantilla que suministra el framework, esto es, a nivel métodos y atributos, y al igual que en actividad anterior, se busca plasmar el código de forma sencilla y entendible a primera instancia. Una vez programa es escrito, se debe probar y ejecutar satisfactoriamente en cluster MPI.

4.3.3. Calcular y analizar valores de métricas de complejidad para ambos algoritmos.

Una vez se obtiene ambos algoritmos se procede a efectuar cálculo de métricas Halstead, Complejidad Ciclomática, LLOC, Índice de Mantenibilidad con base en el código fuente de ambas versiones del algoritmo e-Science, versión MPI y versión Framework. Los cálculos de métricas de Maurice Halstead se apoyan en la cuenta de operadores y operandos del código fuente. El mismo Halstead pública fórmulas para estimaciones de métricas como volumen de código, tiempo aproximado requerido para programar, esfuerzo de programación, dificultad de entendimiento, nivel del programa, y otros [22]. McCabe por su lado expone sobre medida de la complejidad de un programa de acuerdo a sentencias de tipo decisión implícitas en el código [45]. Las líneas de código son otras métricas útiles para determinar la complejidad de una pieza software, inclusive se argumenta que llega a ser la de mayor precisión de las todas las comúnmente utilizadas [44]. Otra métrica utilizadas

para determinar la complejidad de un programa es el índice de mantenibilidad, que combina métricas de McCabe, Halstead y líneas de código [43] y es implementada para evaluar complejidad de esqueletos algorítmicos [31] [41]. Con estas métricas se busca realizar comparativas entre ambos códigos respecto a la premisa de que código más abstracto implica menor esfuerzo, dificultad o complejidad de programar que uno menos abstracto o de menor nivel.

Posterior a consecución de métricas, se realiza paralelo entre las diversas métricas obtenidas de ambas versiones del algoritmo e-Science. Con base en ello se concluye si hipótesis se cumple para este experimento, y que condicionantes o factores existen para que se cumpla o no dicha premisa, y de forma de similar se obtiene respuesta a la pregunta de investigación planteada al inicio de este trabajo.

CAPÍTULO V: PRESENTACIÓN Y ANÁLISIS DE RESULTADOS

En este capítulo del trabajo investigativo se presenta y exponen análisis correspondientes a la ejecución de todas las actividades de los objetivos específicos. Se aborda de manera secuencial los objetivos específicos y sus respectivas actividades.

5.1. CREAR AMBIENTE DE DESARROLLO Y EJECUCIÓN A NIVEL HARDWARE Y SOFTWARE DE ALGORITMOS E-SCIENCE QUE REQUIEREN HPC EN LA UTB

Para que cualquier pieza y pruebas software puedan tener lugar, estas requieren de entornos hardware y software que posibiliten su existencia. En las actividades de este primer objetivo específico, se describe y analiza procedimientos de diseño, análisis, implementación, pruebas, despliegue de ciber infraestructura hardware y software que hagan veces de plataforma para el desarrollo y ejecución de algoritmos o tareas que requieren de HPC. Para ello, se propone actividades en las que se documenta la arquitectura software del ecosistema, y dicha arquitectura se implementa de manera parcial, por medio de instalación y configuración de clusters que emplean tecnologías como MPI, HTCondor y Munin. A continuación se describe y analiza la ejecución de dichas actividades.

5.1.1. Diseñar y Describir arquitectura del sistema

La arquitectura diseñada del sistema se compone como sigue:

5.1.1.1 Requerimientos Funcionales

- Compilación de código fuente en C++
- Ejecución de ejecutables C++
- Creación de archivos de código fuente C++
- Edición de Archivos de código fuente C++
- Eliminación de archivos de código fuente C++
- Monitoreo de actividad de red
- Navegación por sistemas de archivo de sistema operativo

Se determinó los casos de uso o requerimientos funcionales básicos que competen a servicios requeridos por los usuarios finales del sistema. El ecosistema propuesto proporciona de manera implícita una gran cantidad de funcionalidades, a parte de las siete aquí mencionadas, sin embargo, para propósitos de documentación de arquitectura y requerimientos de usuarios listar un gran número de funcionalidades desenfocada de las necesidades esenciales del usuario, contradice principio de la arquitectura software de cualquier sistema.

5.1.1.2. Atributos de Calidad

- Desempeño
- Disponibilidad
- Seguridad
- Mantenibilidad
- Portabilidad

Asimismo, se determinaron características de calidad que se debe satisfacer y que afectan el comportamiento, diseño del sistema y experiencia final del usuario. El desempeño del sistema es importante ya que se desea garantizar alta capacidad y disponibilidad de procesamiento, alto throughput, buenos tiempos de respuesta, alta capacidad acceso a memoria RAM. La propiedad de disponibilidad se prioriza en sistema, ya que se busca proporcionar alta medida de tiempo en servicio y baja tasa de fallos de comunicación. Se prioriza seguridad en el sistema, y se enfatiza esto a nivel de autenticación. La mantenibilidad del sistema se aborda a través del grado de modificabilidad que este permita hacerle. La portabilidad del sistema es vital y se busca garantizar con alto grado de actualización, adaptabilidad y soporte de múltiples lenguajes de programación. Finalmente, la usabilidad se busca garantizar

a través del suministro de mecanismos de accesibilidad al sistema por parte del usuario final.

5.1.1.3. Escenarios Generales y Críticos de Atributos de Calidad

- Disponibilidad
- Modificabilidad (Mantenibilidad)
- Desempeño/Accesibilidad (Usabilidad)
- Autenticación (Seguridad)

Estos escenarios generales y críticos se establecieron con miras a caracterizar requerimientos específicos de los atributos de calidad. Se identifica que la disponibilidad requiere de caracterización de comandos en el sistema, a nivel modificabilidad, se especificaron las consecuencias de cuando se modifica un artefacto del sistema. En cuanto a desempeño, se caracteriza el procesamiento de comandos enviados por el usuario final, y a nivel autenticación, se desea controlar el registro de los usuarios finales al sistema.

5.1.1.4. Aproximaciones Arquitectónicas

Tipo de Aplicación: Web y Escritorio

Debido a la necesidad de hacer uso de recursos computacionales distribuidos, se optó por proporcionar los servicios de HPC a través de una aplicación centrada en interface web e interfaces de escritorio en un nodo centralizado.

Estilos y Patrones Arquitectónicos: Sistemas Distribuidos, Llamado-Retorno; Cliente-Servidor, Orientación a Objetos

Se eligieron estos estilos y patrones con el fin de garantizar metas de calidad tales como capacidad de modificación, desempeño, disponibilidad, portabilidad, usabilidad y seguridad.

Aspectos Transversales: Desempeño, Disponibilidad

Se determinó que estos atributos son transversales puesto que a nivel de módulos y componentes software y hardware, existen lazos de dependencia que influyen en la operación global del sistema.

Tecnologías: Máquinas Virtuales, Sistema Operativo Linux, Compiladores, Paso de Mensajes, Acceso Remoto, Redes de Alta Velocidad, Monitor de Red, Framework de Software

Las tecnologías escogidas satisfacen necesidades tales como:

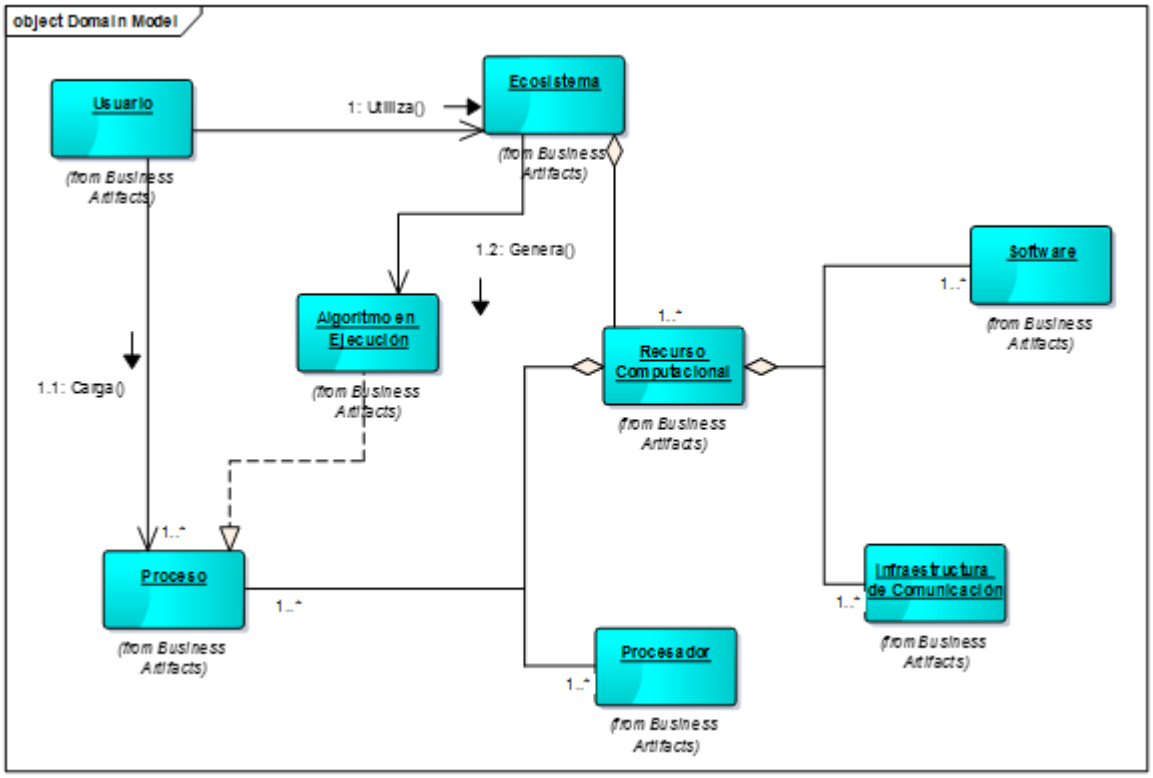
- Alta capacidad de procesamiento
- Protección de datos privados
- Operación ininterrumpida
- Escalabilidad de componentes y módulos software y hardware
- Fácil acceso a tecnología de paso de mensajes
- Identificación de fallas

5.1.1.5. Vistas Del Sistema

5.1.1.5.1. Vista de Negocio

Modelo de Dominio

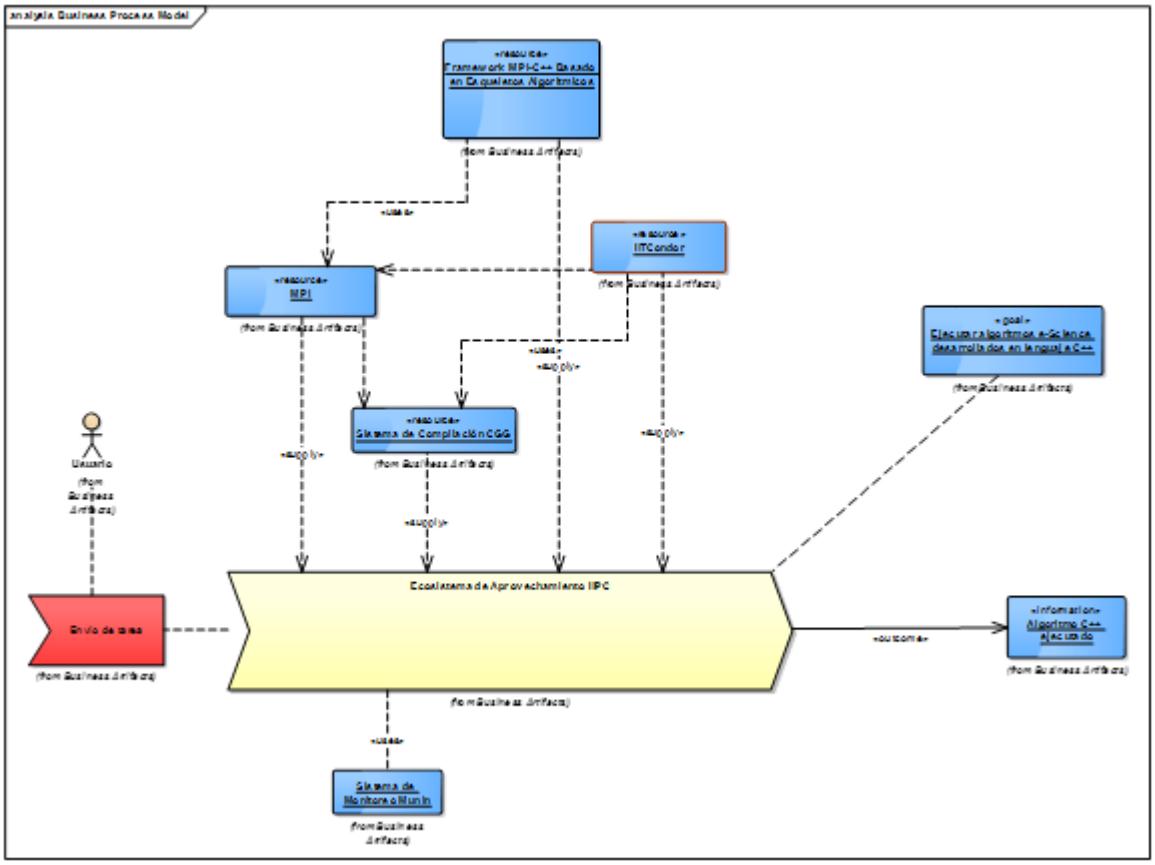
Figura 3: Modelo de Dominio



El modelo de dominio exhibe vista de alto nivel del sistema mediante la definición de objetos físicos y abstractos. De acuerdo a Figura 3 el usuario del sistema utiliza el ecosistema de aprovechamiento HPC, al cual carga uno o varios procesos. El ecosistema genera un algoritmo en ejecución (proceso). Los procesos hacen uso de uno o más procesadores y otros recursos computacionales.

Modelo de Negocio

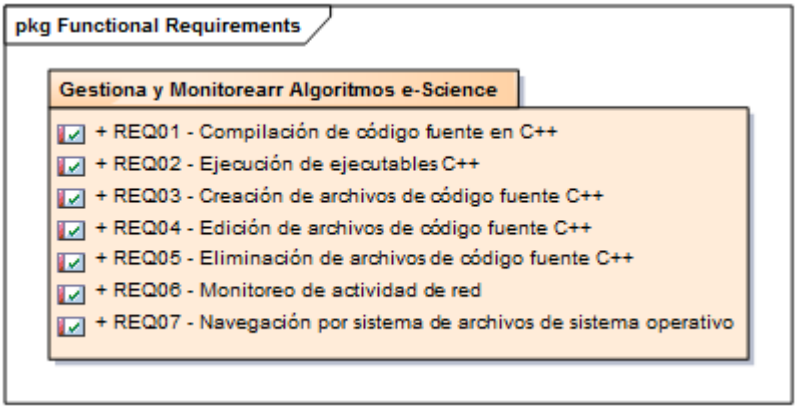
Figura 4: Modelo de Negocio



El modelo de negocio mostrado en la Figura 4, describe, similar al modelo de dominio, vista general o flujo de información dentro del sistema. El usuario final del sistema envía una tarea al ecosistema de aprovechamiento. Dicha tarea es intervenida por subsistemas como MPI, sistema de compilación GCC, HTCondor, Framework inspirado en esqueletos algorítmicos, herramienta de monitoreo de red; al final del proceso, se obtiene como salida la ejecución de la tarea.

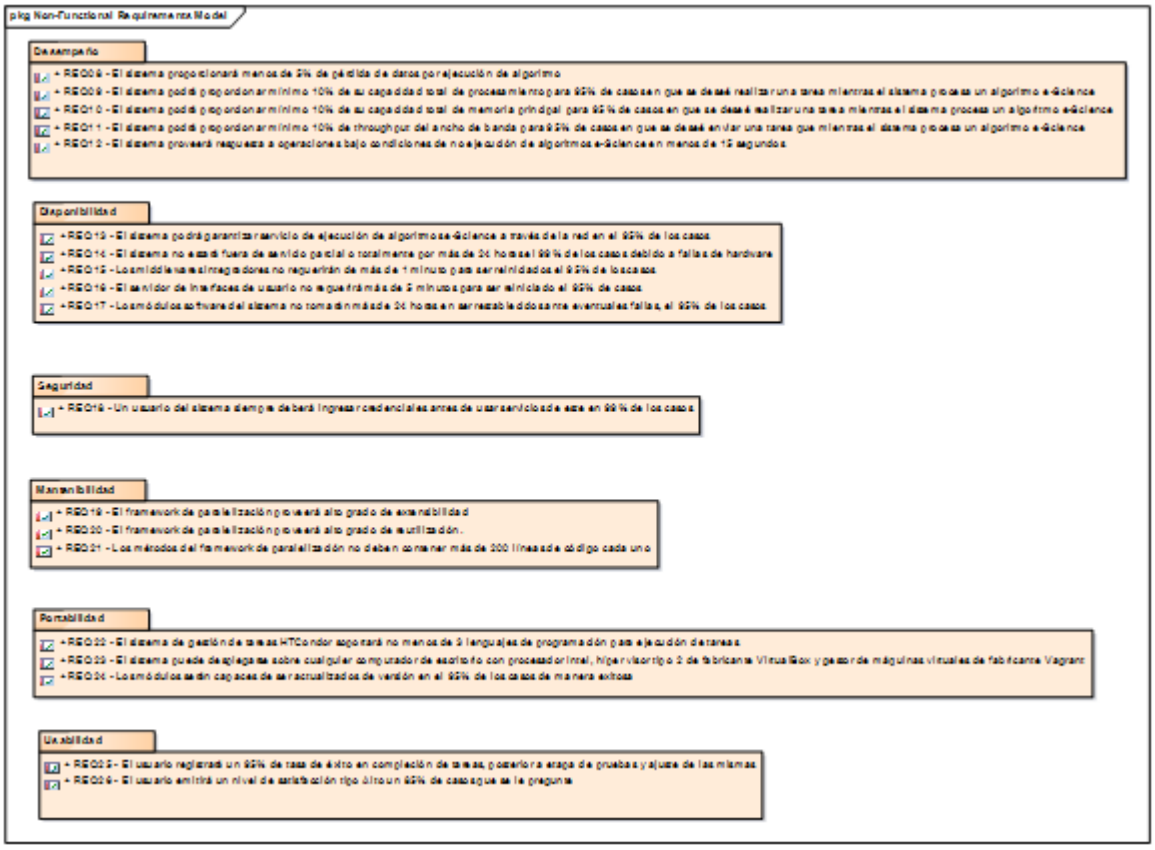
Modelo de Requerimientos

Figura 5: Requerimientos Funcionales



El modelo de requerimientos funcionales agrupa los casos de uso básicos que requiere el usuario final del sistema. La Figura 5 exhibe las siete funciones básicas que el ecosistema de aprovechamiento HPC proporciona a sus usuarios finales.

Figura 6: Requerimientos No Funcionales

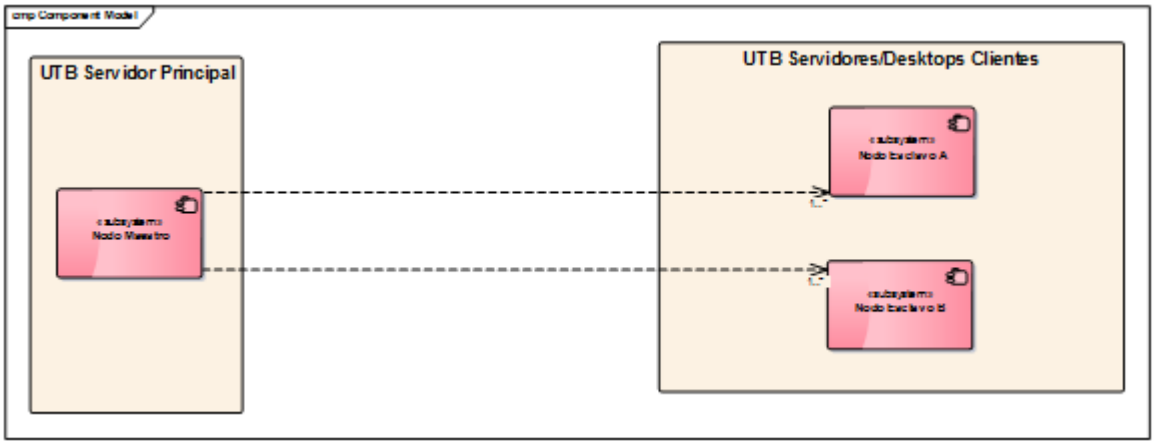


El modelo de requerimientos no funcionales agrupa los atributos de calidad y requerimientos básicos que el sistema debe proporcionar. La Figura 6 exhibe los dieciocho requerimientos no funcionales que el ecosistema de aprovechamiento HPC busca satisfacer.

5.1.1.5.2. Vista Lógica

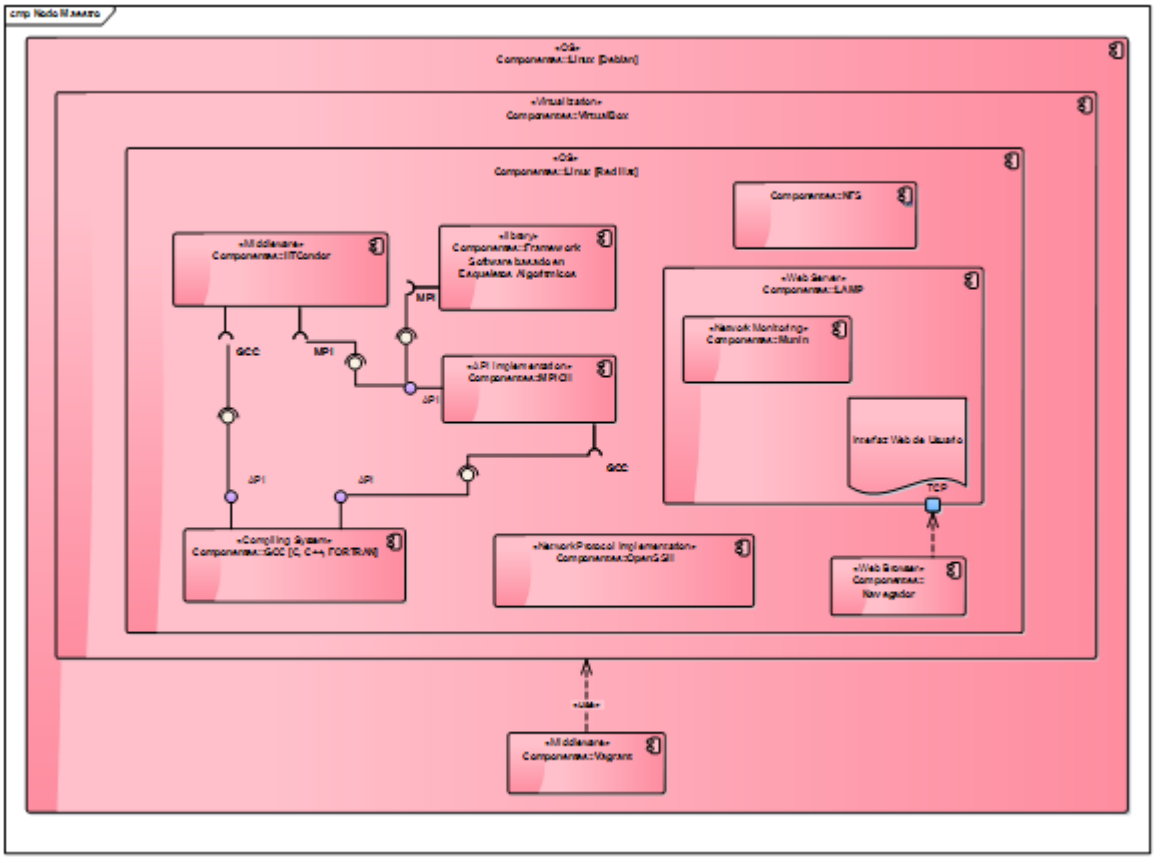
Modelo de Componentes

Figura 7: Modelo de Componentes



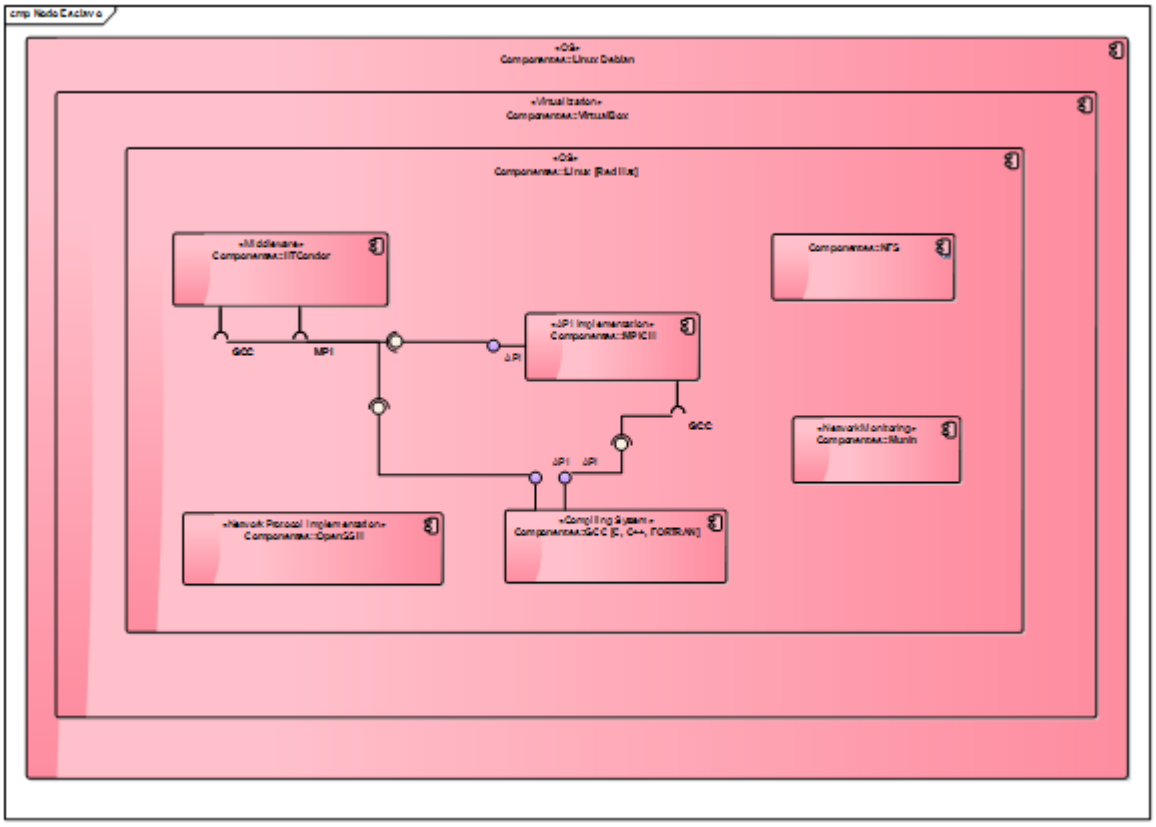
El modelo de componentes de la arquitectura presenta las piezas software, interfaces y dependencias que componen el sistema. La Figura 7 exhibe el componente Nodo Maestro, que conecta con dos tipos de Nodo Esclavo, tipo A y B, y que pueden ser uno o muchos de cada uno. Estos componentes ejecutan en distintos entornos de despliegue descritos en modelo de despliegue. Estos tres tipos de componentes tienen una estructura interna que se detalla a continuación.

Figura 8: Modelo de Componente Maestro



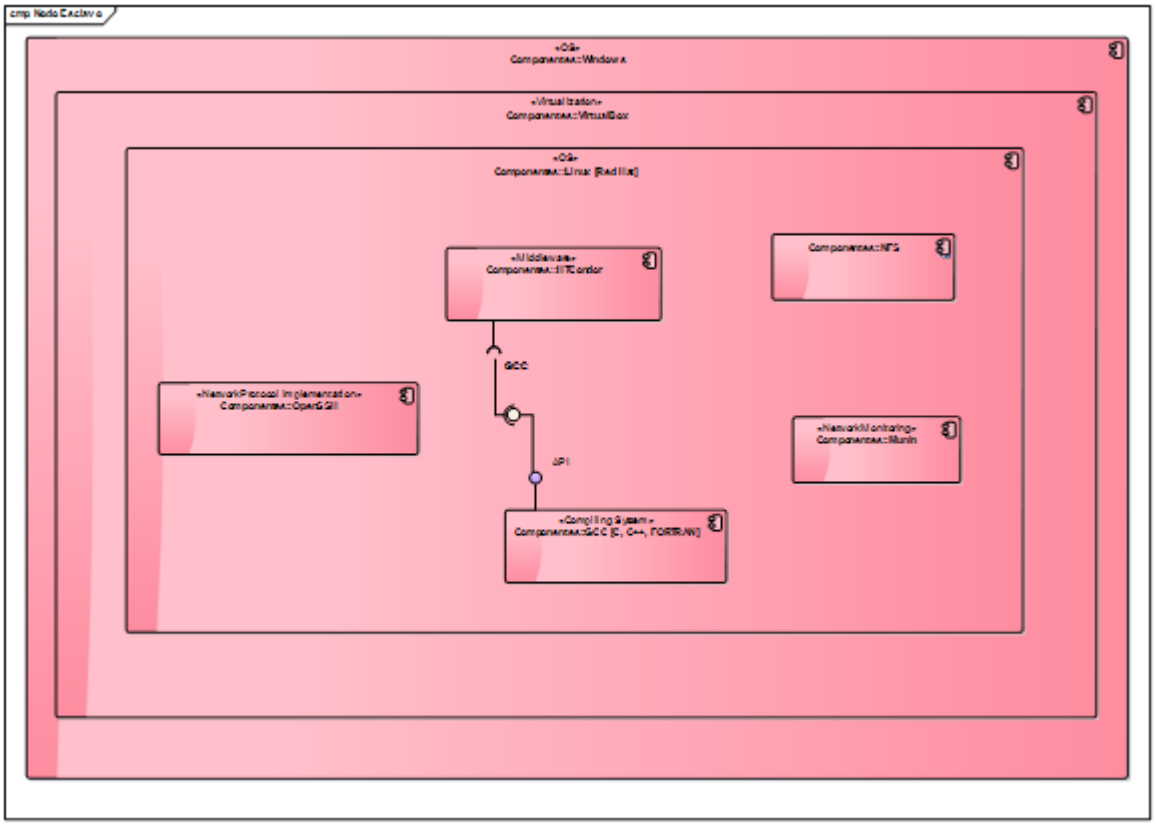
El componente Nodo Maestro, de acuerdo con Figura 8, detalla las interconexiones, dependencias, interfaces y componentes internos que interactúan para suministrar servicios (requerimientos funcionales y no funcionales) en el marco del ecosistema de aprovechamiento. El nodo opera bajo un entorno Linux, que a su vez ejecuta una máquina virtual Linux, la cual contiene paquetes y servicios software como HTCondor, MPI, GCC, SSH, Munin, NFS, Vagrant.

Figura 9: Modelo de Componente Esclavo A



El componente Nodo Esclavo tipo A de acuerdo con Figura 9, detalla las interconexiones, dependencias, interfaces y componentes internos que interactúan para suministrar servicios (requerimientos funcionales y no funcionales) en el marco del ecosistema de aprovechamiento. El nodo opera bajo un entorno Linux, que a su vez ejecuta una máquina virtual Linux, la cual contiene paquetes y servicios software como HTCondor, MPI, GCC, SSH, Munin, y NFS. Si bien su estructura interna es muy similar a la del componente Nodo Maestro, los detalles de despliegue de los mismos varía.

Figura 10: Modelo de Componente Esclavo B

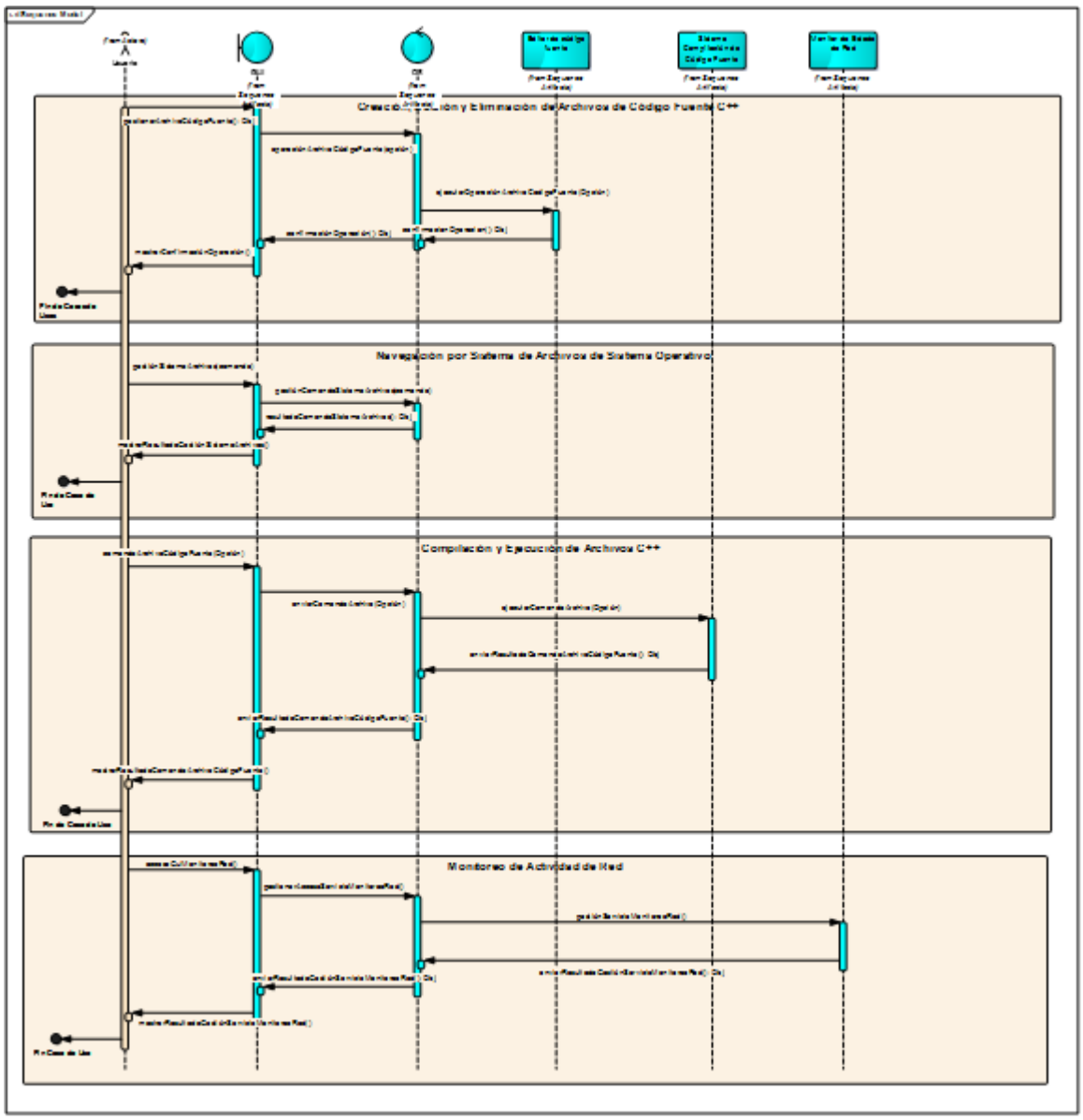


El componente Nodo Esclavo tipo B, de acuerdo con Figura 10, detalla las interconexiones, dependencias, interfaces y componentes internos que interactúan para suministrar servicios (requerimientos funcionales y no funcionales) en el marco del ecosistema de aprovechamiento. El nodo opera bajo un entorno Linux, que a su vez ejecuta una máquina virtual Linux, la cual contiene paquetes y servicios software como HTCondor, MPI, GCC, SSH, Munin, y NFS. Si bien su estructura interna es muy similar a la del componente Nodo Esclavo tipo B y el Maestro, sus detalles de despliegue varían.

5.1.1.5.3. Vista de Procesos

Modelo de Secuencia

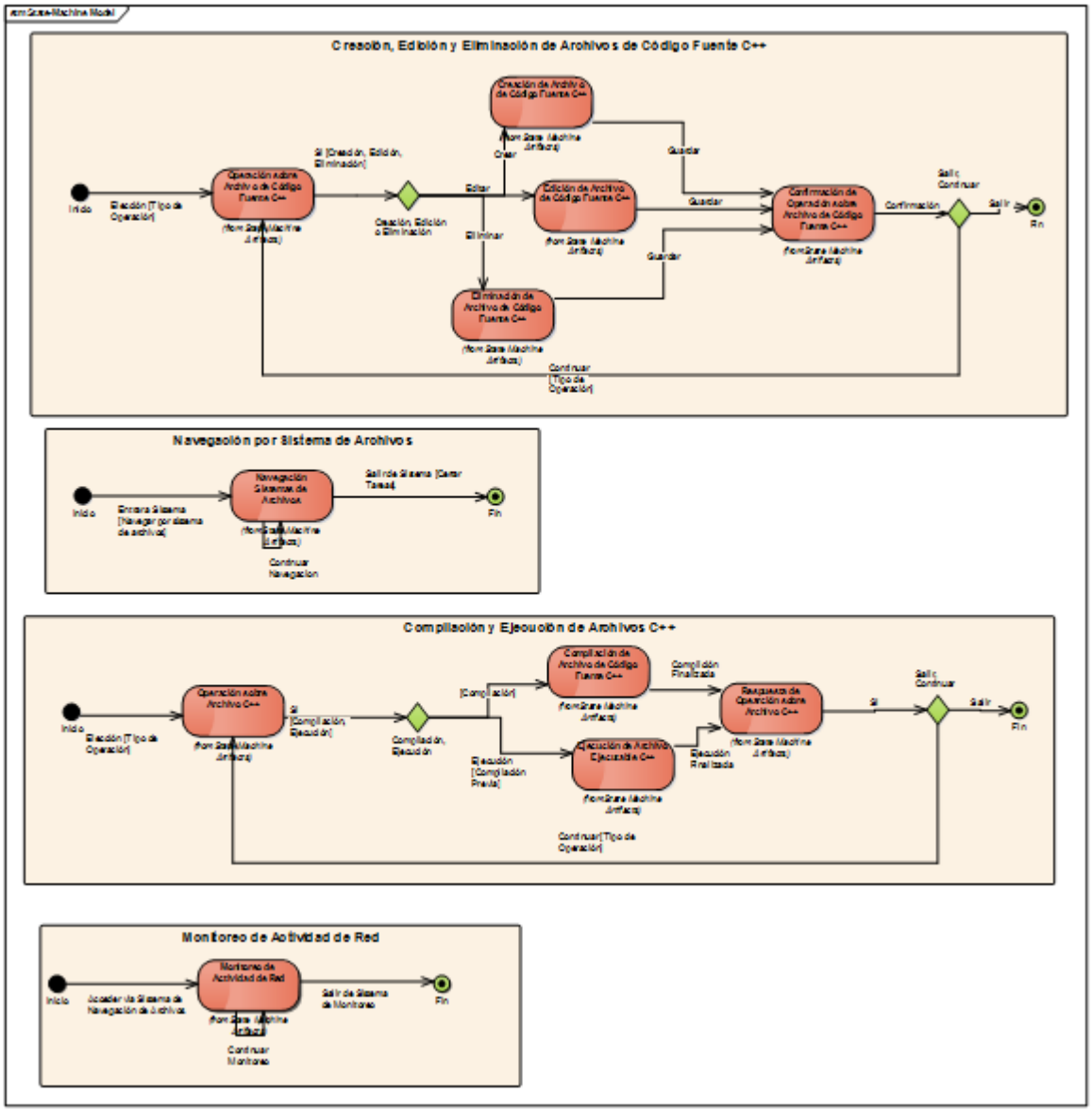
Figura 11: Modelo de Secuencia



El modelo Secuencia es aquél donde el sistema es descrito acorde a comportamientos funcionales que experimenta el sistema. Este comportamiento se describe en términos de mensajes secuenciales a través de múltiples objetos del sistema. En la Figura 11 se exhibe diagrama de secuencia que corresponde a los requerimientos funcionales expuestos en la arquitectura del sistema.

Modelo de Máquina-Estado

Figura 12: Modelo de Máquina-Estado



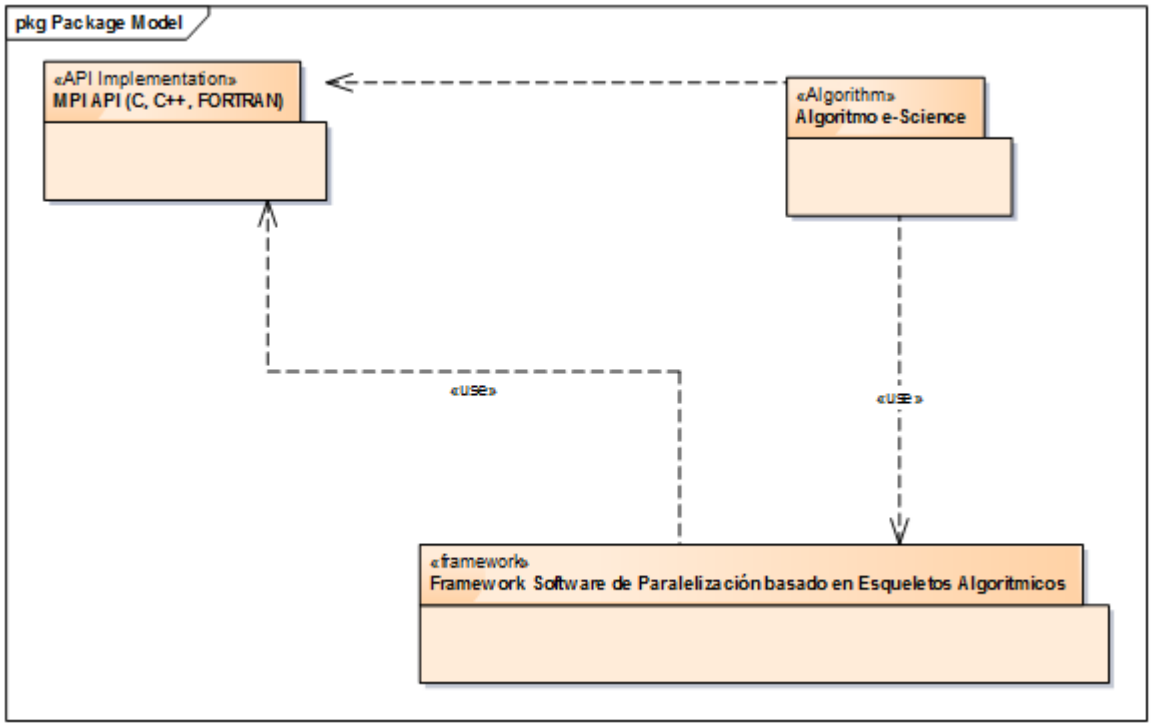
El modelo Máquina-Estado es aquél donde el sistema es ilustrado acorde con comportamientos clasificados en estados, que cambian a través de transiciones que se rigen bajo restricciones y desencadenantes. En la Figura 12 se exhibe diagrama

Máquina-Estado que corresponde a los requerimientos funcionales expuestos en la arquitectura del sistema.

5.1.1.5.4. Vista de Implementación

Modelo de Paquetes

Figura 13: Modelo de Paquetes

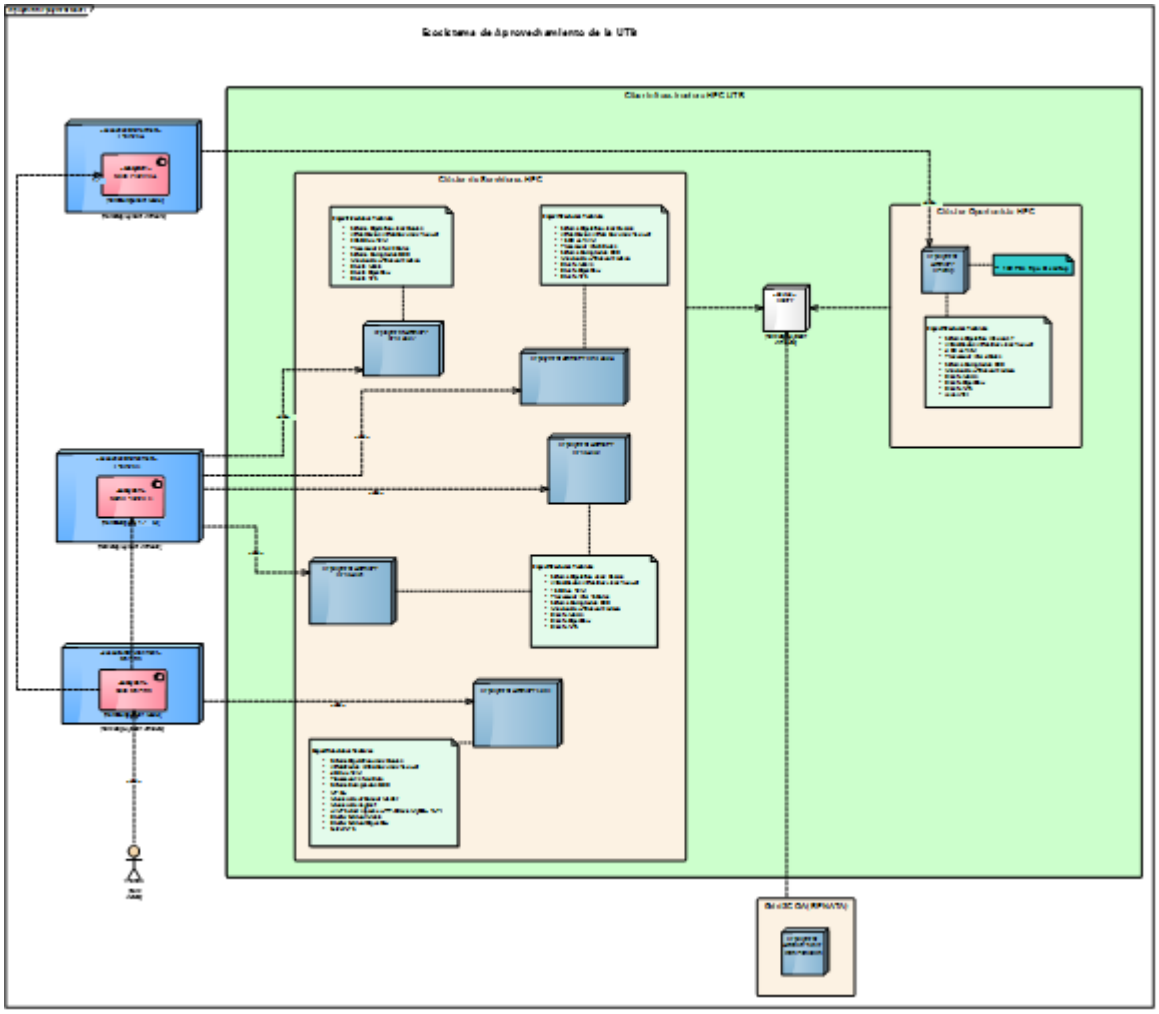


El modelo de Paquetes del sistema software del ecosistema es un diagrama que expresa la composición y dependencias de un sistema software a alto nivel por medio de unidades estructurales como lo son los paquetes. La Figura 13 muestra que el framework de paralelización utiliza y depende de la librería MPI, mientras que

el algoritmo e-Science desarrollado por el usuario final utiliza o se adapta al framework de paralelización.

5.1.1.5.5. Vista de Despliegue

Figura 14: Modelo de Despliegue



El modelo de despliegue, como el que se aprecia en la Figura 14, es aquél donde se especifican entorno de ejecución, dónde y cómo el sistema se despliega a nivel físico. El ecosistema de aprovechamiento HPC se segmenta en tres espacios, cluster de servidores, cluster laboratorios de cómputo y la grid Renata. El clúster HPC de servidores consta de 5 servidores de alta capacidad de procesamiento y memoria principal, sobre el cual el usuario final tiene acceso al sistema. Esta red se enlaza con cluster oportunista de laboratorio de redes, el cual se diseña para alrededor de 150 PCs de escritorio, los cuales proporcionan capacidad de procesamiento en segundo plano, en esquema oportunista. Adicionalmente, el usuario puede contar con capacidad de procesamiento de respaldo gracias a enlace del ecosistema de aprovechamiento con grid HPC de 3COA de Renata, la cual incorpora clusters de computadores ubicados en varios centros investigativos de Colombia.

Para mayor detalle sobre la arquitectura del sistema referirse a anexo 01. A continuación, se reporta sobre materialización parcial de la arquitectura, donde ello se logra mediante la implementación del cluster con integración de tecnologías MPI, HTCondor y de monitoreo de actividad de red, con el objetivo de poner en marcha un entorno de desarrollo y ejecución de tareas e-Science que requieren HPC.

5.1.2. Configurar cluster MPI

La configuración de cluster MPI requirió considerar como mínimo dos esquemas de despliegue de ciber infraestructura. Por un lado, un esquema para un Nodo Maestro y otro para los Nodos Esclavos. La arquitectura del sistema contempla la clonación de nodos esclavos mediante tecnologías de máquinas virtuales, por lo que se requiere de dos esquemas de configuración, que para momento de clonación de nodos esclavos requiere de leves ajustes adicionales. A continuación, se detalla resultados de configuración de Nodo Maestro de cluster MPI.

5.1.2.1. Nodo Maestro

Para empezar, CentOS, en su versión 32 bits, se instaló sobre una máquina virtual Linux de memoria principal 2GB y con capacidad inicial de disco duro de 10GB y se le asigna un núcleo de procesamiento. La distribución Linux CentOS deriva de la distribución Red Hat Enterprise Linux y proporciona variados tipos de servicios, entre estos, paquetes y tecnologías que posibilitan la creación de clusters para HPC. Se eligió una instalación definitiva del sistema operativo, ya que se requiere almacenar de forma permanente archivos de ejecución, desarrollo, configuración entre otros, para satisfacer los múltiples requerimientos funcionales y atributos de calidad del sistema.

Se estableció un usuario administrador en el Nodo Maestro con el propósito de gestionar acceso y acciones como instalación, desinstalación, configuración de paquetes software en el nodo, creación, eliminación y configuración de usuarios, entre otras. También se creó y configuró un usuario adicional, con menos privilegios administrativos que el usuario raíz del Nodo Maestro, pero que permite a su vez la configuración y uso de diferentes servicios que requieren de bajos niveles de acceso de seguridad, y facilita el ajuste de servicios de almacenamiento por red y acceso remoto sin contraseña, los cuales satisfacen atributos como el desempeño, seguridad, usabilidad y mantenibilidad.

Por otro lado, para el caso del Nodo Maestro, se estableció la cadena ‘king’ como ejemplo de hostname para el nodo. El hostname es una cadena de caracteres que identifica al nodo en su red de cómputo desde punto de vista de las aplicaciones. Adicionalmente, se registró cada hostname y dirección IP asociada de la red de cómputo en el archivo ‘hosts’ del nodo. En futuras configuraciones de paquetes software, estos pueden hacer uso de este archivo para mapear los hosts del cluster, de manera que se haga sencillo para el usuario final conocer los nodos a los que cada nodo puede hacer contacto por red.

También, se deshabilitó el módulo de seguridad por Firewall y Security Enhance Linux para minimizar restricción a acceso de recursos compartidos, tales como archivos. Si bien se compromete en cierto grado la seguridad del sistema al deshabilitar ambos servicios, estas se compensan con otras medidas, y también es

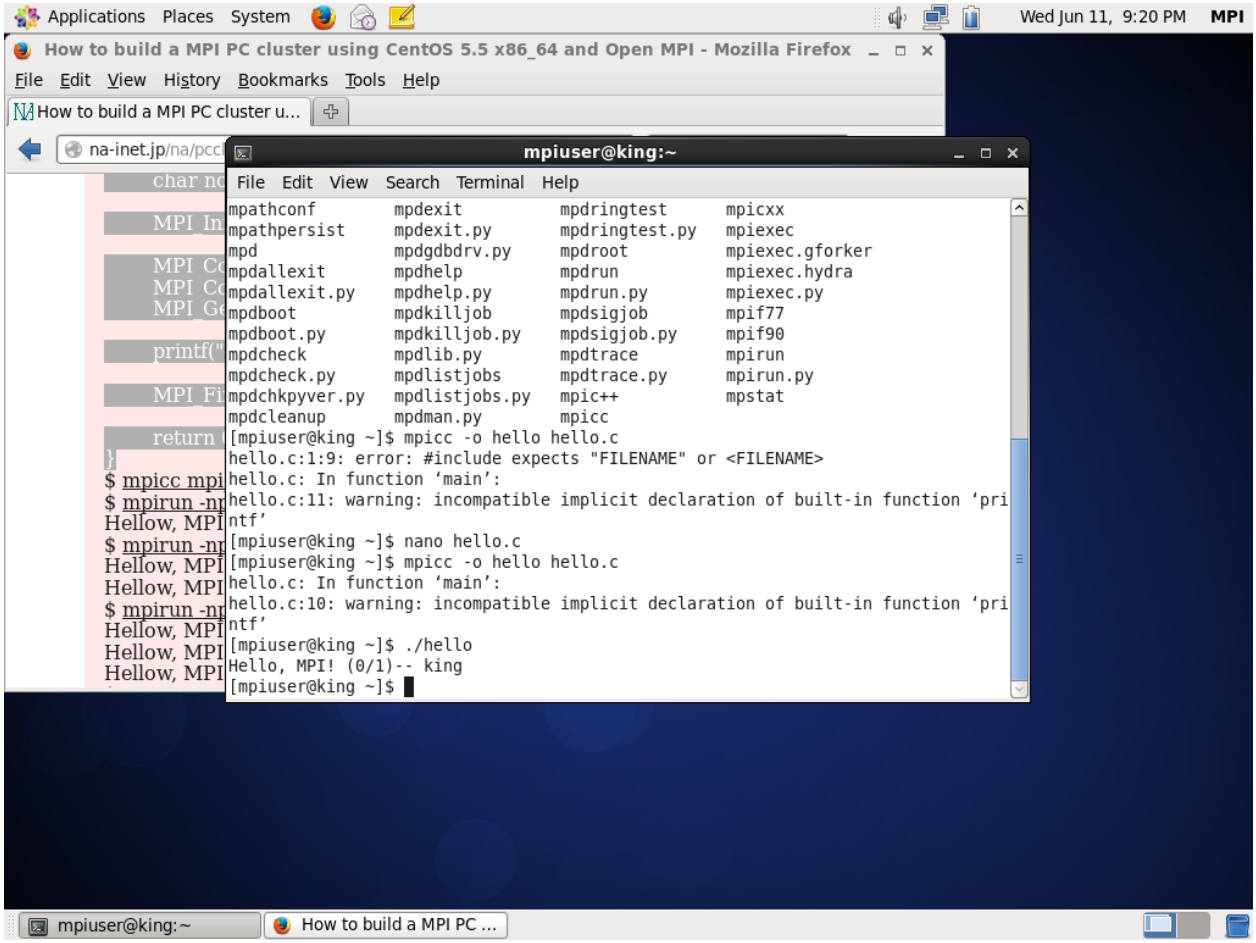
importante resaltar que atributos como desempeño y disponibilidad tienen prioridad sobre el de seguridad.

Se configuró el protocolo Network File System (NFS) para habilitar almacenamiento masivo distribuido por red, con el objetivo de minimizar el tamaño de las máquinas virtuales de los Nodos Esclavos, y así hacerlas lo más portables posibles. Adicionalmente, se satisfacen otros requerimientos no funcionales. El Nodo Maestro hace veces de servidor NFS (almacena físicamente los archivos de la red) y para ello, se publica él, o los directorios que los demás nodos del cluster acceden por red, sin mayores restricciones de seguridad.

Se descargó y realizó instalación básica de paquetes implementadores de SSH versión servidor y cliente, y MPI. SSH es un protocolo de comunicaciones que permite acceder un nodo a otro a través de la red y tomar control de sus capacidades, ello con fin de facilitar labores de comunicaciones entre los nodos del cluster. Por otro lado, MPI es la interfaz que permite compilar y ejecutar código paralelo en clusters HPC y satisface atributos de calidad como desempeño, disponibilidad y portabilidad, y algunos requerimientos funcionales del ecosistema.

Se configuró una clave compartida para dar acceso remoto a nodos del cluster a través de protocolo SSH. Esta configuración se realizó con miras a que, al enviar tareas al cluster, no sea necesario que para acceder a recursos de procesamiento remotos, se pida credenciales de acceso a los diferentes nodos del cluster, satisfaciendo también así necesidades de desempeño..

Figura 15: Prueba de MPI con Nodo Master



Para finalizar el tema configuraciones en el Nodo Maestro, se realizó compilación y ejecución de un algoritmo e-Science escrito en código C, con uso de un sólo nodo del cluster, el Maestro en este caso. Figura 15 muestra salida que produce ejecución de un algoritmo MPI escrito en código C, con uso solamente del Nodo Maestro del cluster.

5.1.2.2. Nodo Esclavo

En los Nodos Esclavos se realizaron los mismos procedimientos que en el Nodo Maestro, esto relacionado a instalación de sistema operativo Linux CentOS, configuración de usuario raíz y usuario no administrativo ‘mpiuser’. De allí en adelante, el proceso de configuración fue similar al del Nodo Maestro con ligeras variaciones que se describen a continuación. Los Nodos Esclavos utilizaron a nivel de especificaciones técnicas, sistema operativo CentOS versión 32 bits, que se instala sobre una máquina virtual Linux de memoria principal 1GB, con capacidad inicial de disco duro de 10GB y se le asigna un núcleo de procesamiento. Adicionalmente, las configuraciones software y hardware realizadas en dichos nodos satisfacen mismo requerimientos funcionales y no funcionales que las llevadas a cabo en el Nodo Maestro.

Por otro lado, para el caso de un Nodo Esclavo, se establece la cadena de caracteres ‘slave1’ como ejemplo de hostname. Adicionalmente, se registró cada hostname y dirección IP asociada de la red de cómputo en el archivo hosts del nodo. En futuras configuraciones de paquetes software, estos pueden hacer uso de este archivo para mapear los hosts del cluster, de manera que sea sencillo para el usuario final conocer los nodos con los que otros nodos realizan transmisiones de datos.

Todo cluster o servicio de cómputo distribuido depende de comunicaciones entre nodos a través de redes, por ello, se configuró adaptador de red bajo protocolo IPv4 para los Nodos Esclavos, esto incluye dirección IP y máscara de subred. Se realizó prueba de conectividad de nodos a través de la red local por medio de comando ping, esto para garantizar conectividad entre los nodos del cluster.

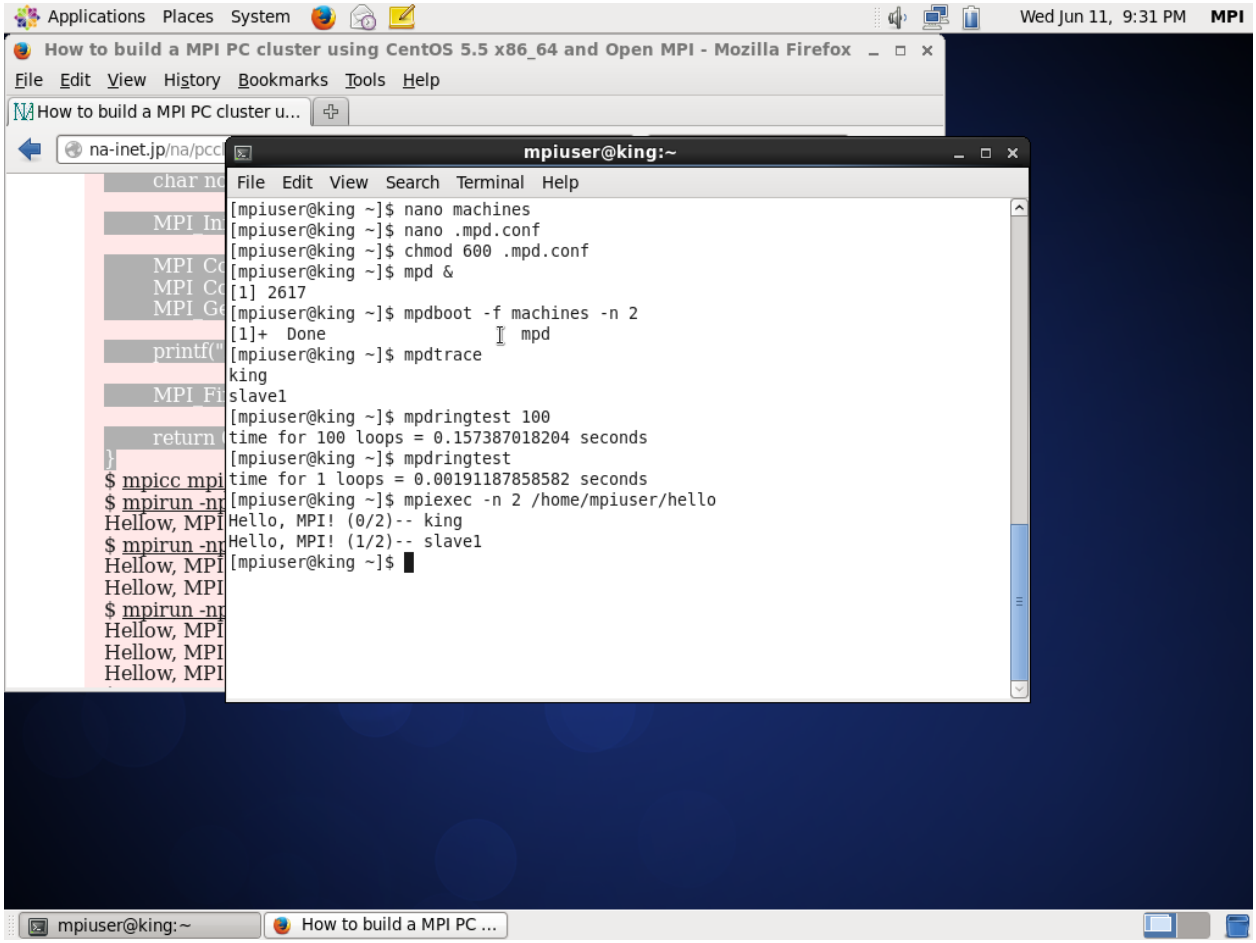
Los Nodos Esclavos registraron los directorios a compartir por protocolo NFS y especificaron al Nodo Maestro como el servidor NFS. El archivo que se intervino fue ‘fstab’, con el cual se buscó configurar sistemas de archivos, NFS en este caso.

Se descargó y realizó instalación básica de paquetes implementadores de SSH versión servidor cliente, y MPI. SSH es un protocolo de comunicaciones que permite acceder un nodo a otro a través de la red y tomar control de sus capacidades,

característica clásica de los clusters HPC. Adicionalmente, MPI es la interfaz que permite compilar y ejecutar código paralelo en dichos clusters HPC.

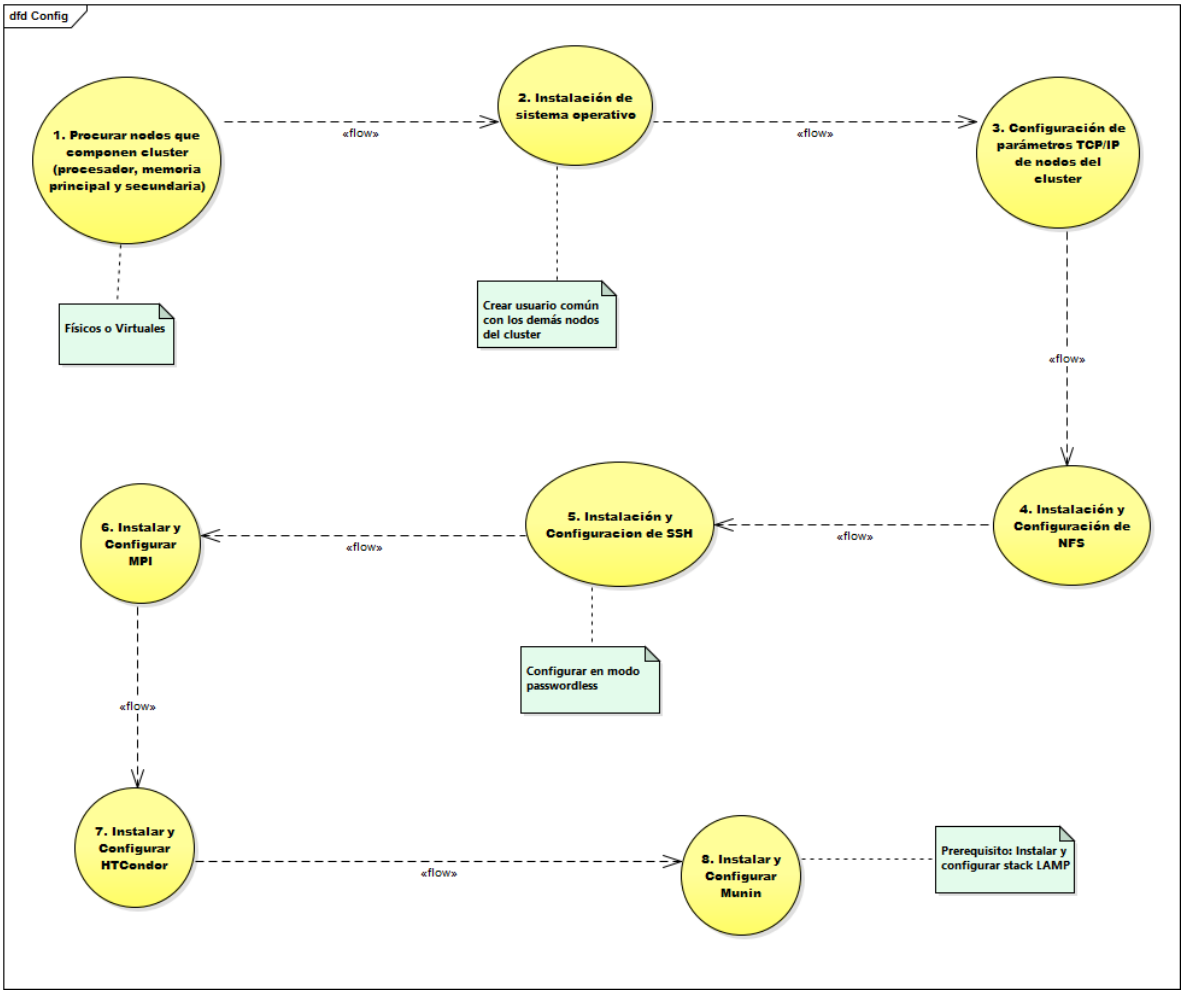
Para insertar un Nodo Esclavo en el cluster MPI, se inicializó proceso MPI en cada esclavo. La implementación MPI escogida, ‘MPICH, requiere para enlazar los nodos del cluster que cada uno de estos inicialice dicho proceso. Adicionalmente, el Cluster MPI requirió de configuraciones adicionales tales como iniciación de procesos MPI, y relación de nodos del cluster a través de sus respectivos hostname. La iniciación del proceso base MPI en cada nodo del cluster es importante para que cada uno de estos ingrese de manera lógica a la red MPI que ensambla el Nodo Maestro. La asignación de los hostname a archivo centralizado que maneja Nodo Maestro fue con motivo de controlar cual o cuales nodos participaban en el procesamiento de las tareas y a que nodo se le asigna determinado rango/ID dentro del cluster. Este archivo centralizado es importante para manipular aspectos de distribución en la composición de algoritmos MPI.

Figura 16: Prueba de MPI con dos nodos



Se realizó prueba de cluster MPI a través de ejecución de algoritmo con sentencias MPI, escrito en código C, con uso de dos nodos del cluster. Luego de las instalaciones y configuraciones realizadas en el Nodo Maestro y un Nodo Esclavo, se confirmaron los correctos procedimientos mediante la ejecución de algoritmos en el cluster. Figura 16 muestra salida que produce ejecutar algoritmo MPI en cluster compuesto por dos nodos. Mientras que la Figura 17 muestra un esquema en alto nivel sobre los pasos que se siguieron para configurar el cluster, ello aplica tanto para un nodo tipo ‘Maestro’ como uno ‘Esclavo’.

Figura 17: Esquema de pasos a seguir para instlar cluster MPI



A continuación, se detallan resultados de configuración de software especial para integrar un cluster MPI con otros clusters, para conformar una grid. Similar al desarrollo de esta actividad, la siguiente actividad se explica con base a procedimientos realizados en el Nodo maestro y los Nodos Esclavos.

5.1.3. Configurar middleware para integrar cluster y grid

Para Nodo Maestro y Nodos Esclavos se realizaron procedimientos similares para integrar Cluster MPI con el servicio grid existente de la red de investigación 3COA de RENATA, por intermedio de middleware HTCondor. Dicho middleware garantiza satisfacción de requerimientos funcionales del ecosistema y de atributos como desempeño, disponibilidad, portabilidad, mantenibilidad, seguridad y usabilidad.

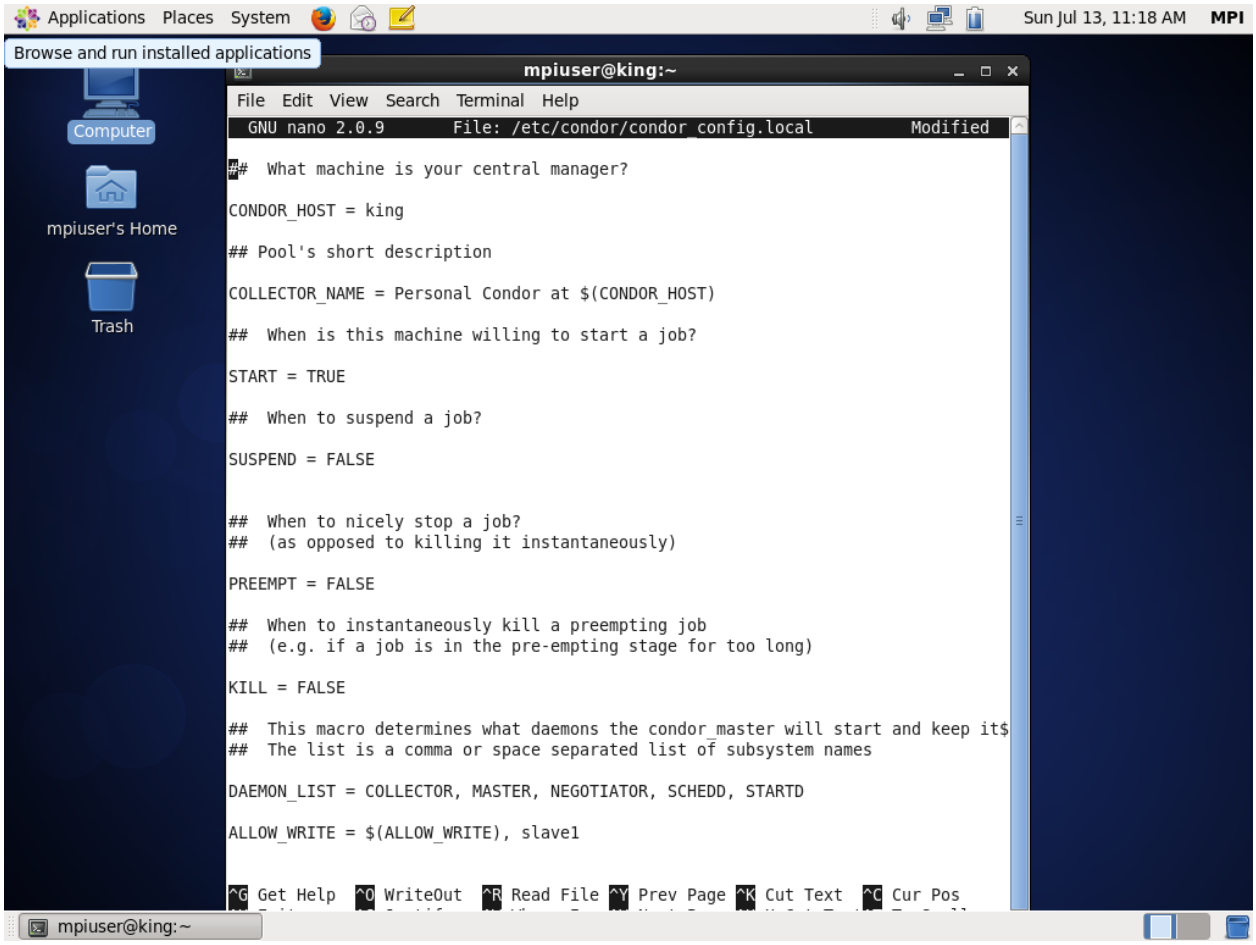
5.1.3.1. Nodo Maestro

Para comenzar, se descargó e instaló el paquete integrador para soluciones de computación de alto desempeño. Las herramientas que proporciona HTCondor permiten compilar y ejecutar tareas en paralelo, por lotes, o seriales, con uso de diferentes lenguajes de programación y sistemas operativos, lo que le hace muy atractivo para utilizar como plataforma HPC común.

HTCondor se explotó al modificar parámetros de sus archivos de configuración. El Nodo Maestro del cluster recibió características diferentes a las de cualquier otro nodo del cluster, ello con el fin de establecer cuales nodos se encargan de procesar y ejecutar código fuente, y cuales nodos gestionan comunicación y otros recursos computacionales entre los nodos del cluster.

Se inició servicio HTCondor en Nodo Maestro, como también se verificó procesos que el middleware ejecuta en segundo plano, con el fin de establecer la correcta configuración del servicio en este nodo. El inicio del servicio se configuró para que arranque al iniciar los nodos, esto con el objetivo de brindar disponibilidad del mismo tan pronto los equipos se encuentren encendidos.

Figura 18: Archivo de configuración HTCondor en Nodo Maestro



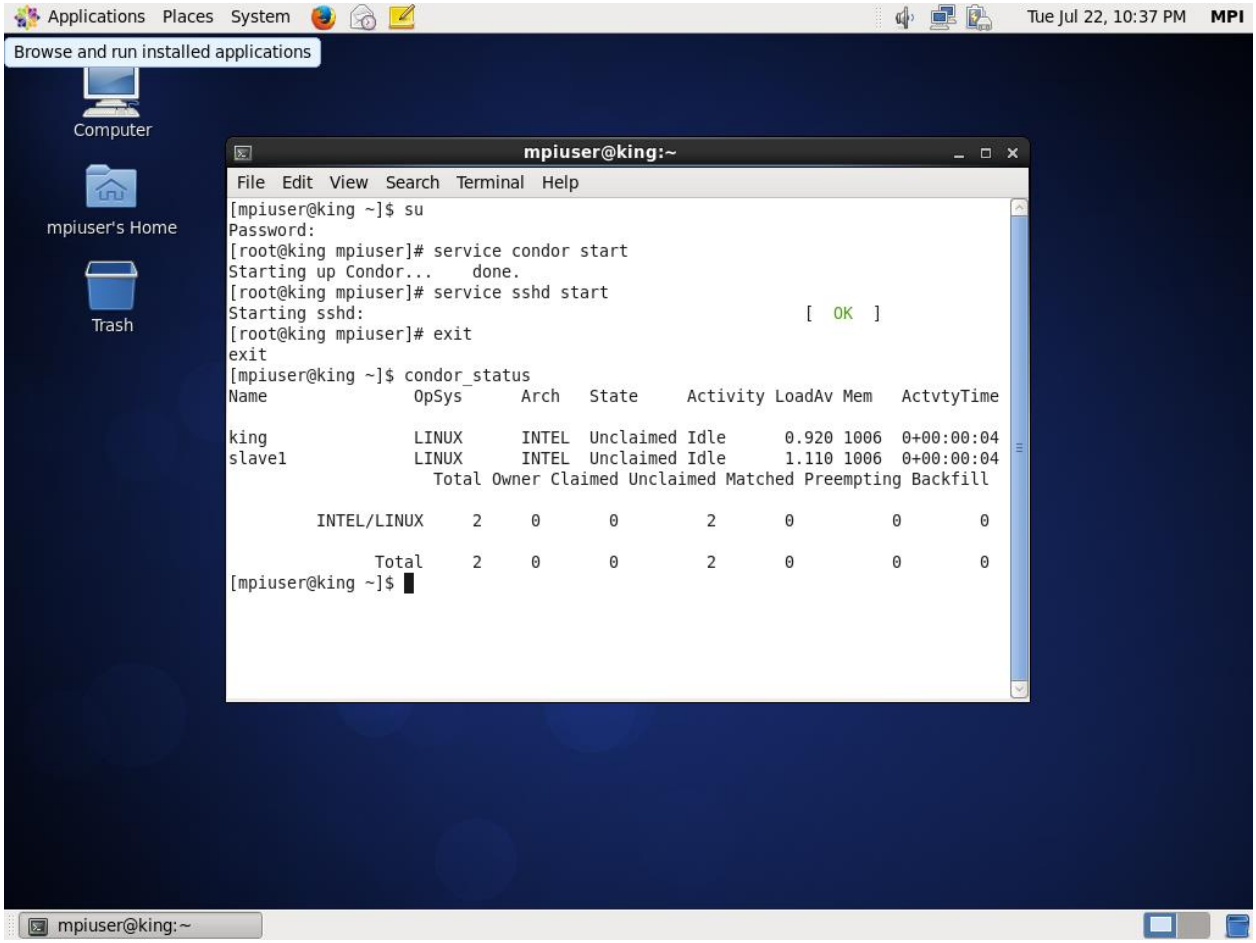
El archivo de configuración, acorde a Figura 18, muestra las responsabilidades del Nodo Maestro del cluster. El Nodo Maestro gestiona procesos en todos los nodos del cluster, ejecuta tareas, recolecta información sobre de estado del cluster, negocia prioridad de tareas dentro del este y puede enviar tareas al mismo.

5.1.3.2. Nodo Esclavo

En los Nodos Esclavos el middleware HTCondor se descargó e instaló de igual manera que se realizó en Nodo Maestro. Asimismo, el archivo de configuración del Nodo Esclavo registra las responsabilidades del Nodo Esclavo del cluster. Los Nodos Esclavos gestionan procesos en todos los nodos del cluster y procesan tareas. También, se les debe designar explícitamente el hostname del nodo del cluster que hace las veces de Nodo Maestro.

Se inició de servicio HTCondor en Nodo Esclavo, y también como se verificó los procesos que el middleware ejecuta en segundo plano.

Figura 19: HTCondor pool corriendo en cluster



Una vez se finalizó ajustes en archivos de configuración, se confirma si nodos del cluster aparecen en pool de máquinas del mismo. Figura 19 muestra pool de máquinas HTCondor del cluster desplegado previamente.

Como se mencionó con anterioridad, HTCondor funcionó para hacer interfaz entre el usuario y otras tecnologías de procesamiento, como lenguajes de programación e interfaces como MPI. En adición, Condor permitió integrar cluster HTCondor para que operara como una grid y se expanda aún más el poder de procesamiento del ecosistema.

5.1.3.3. Grid Existente

La Universidad Tecnológica de Bolívar tiene acceso a nodos de procesamiento de la red RENATA a nivel Colombia. Dicha grid es accedida por medio de integración con middleware HTCondor. El servidor/supercomputador grid de la UTB tiene instalado y configurado HTCondor para enlazar con otros servidores HTCondor y procesar tareas en estos, para el caso en que recursos HPC locales estén ocupados y los remotos tengan disponibilidad para procesar tareas. Dicho servidor grid de la UTB también hace parte del Cluster MPI. A continuación, se expone estado de configuración de HTCondor para acceder a recursos grid externos.

Figura 20: Acceso a ciber infraestructura HPC previa de UTB

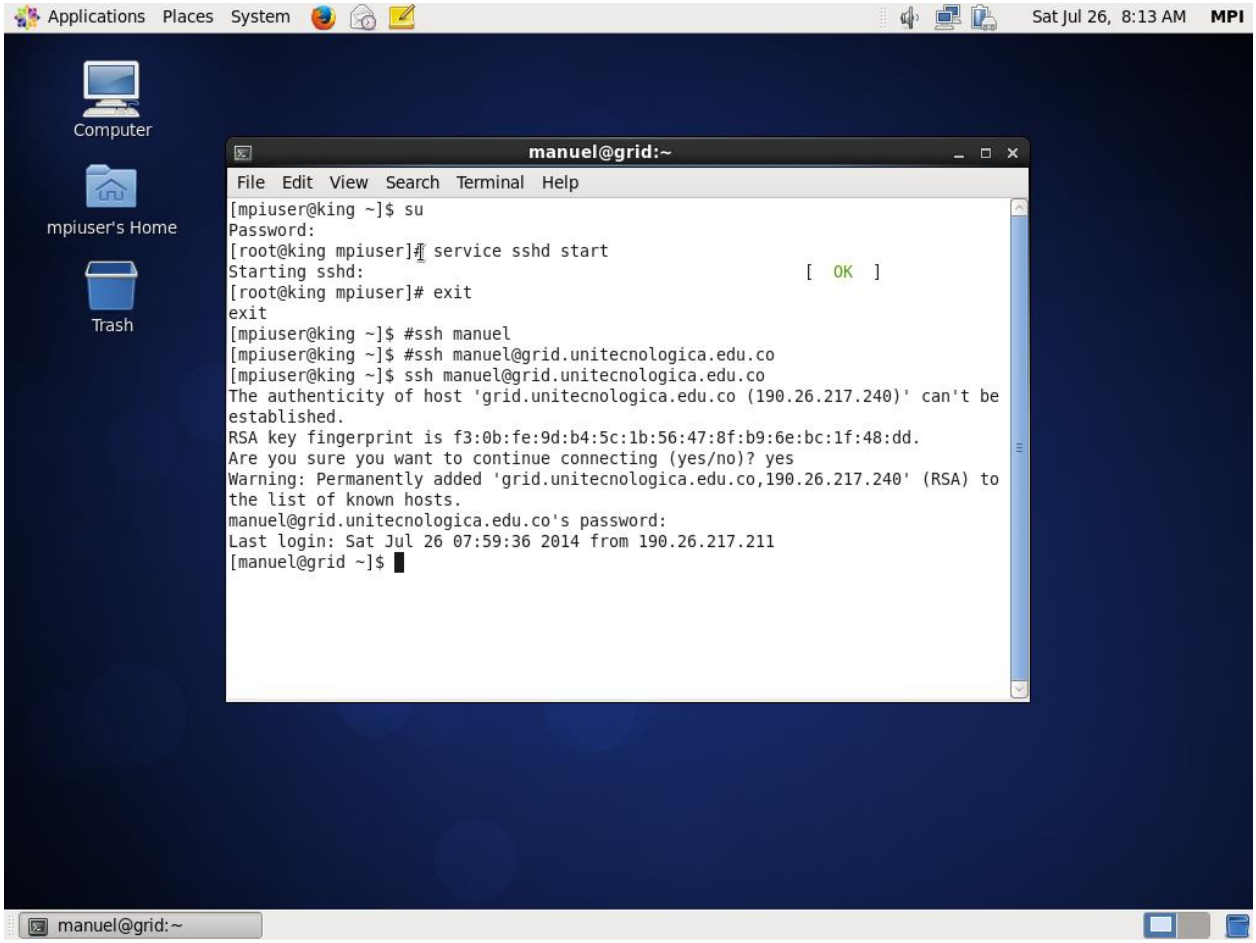
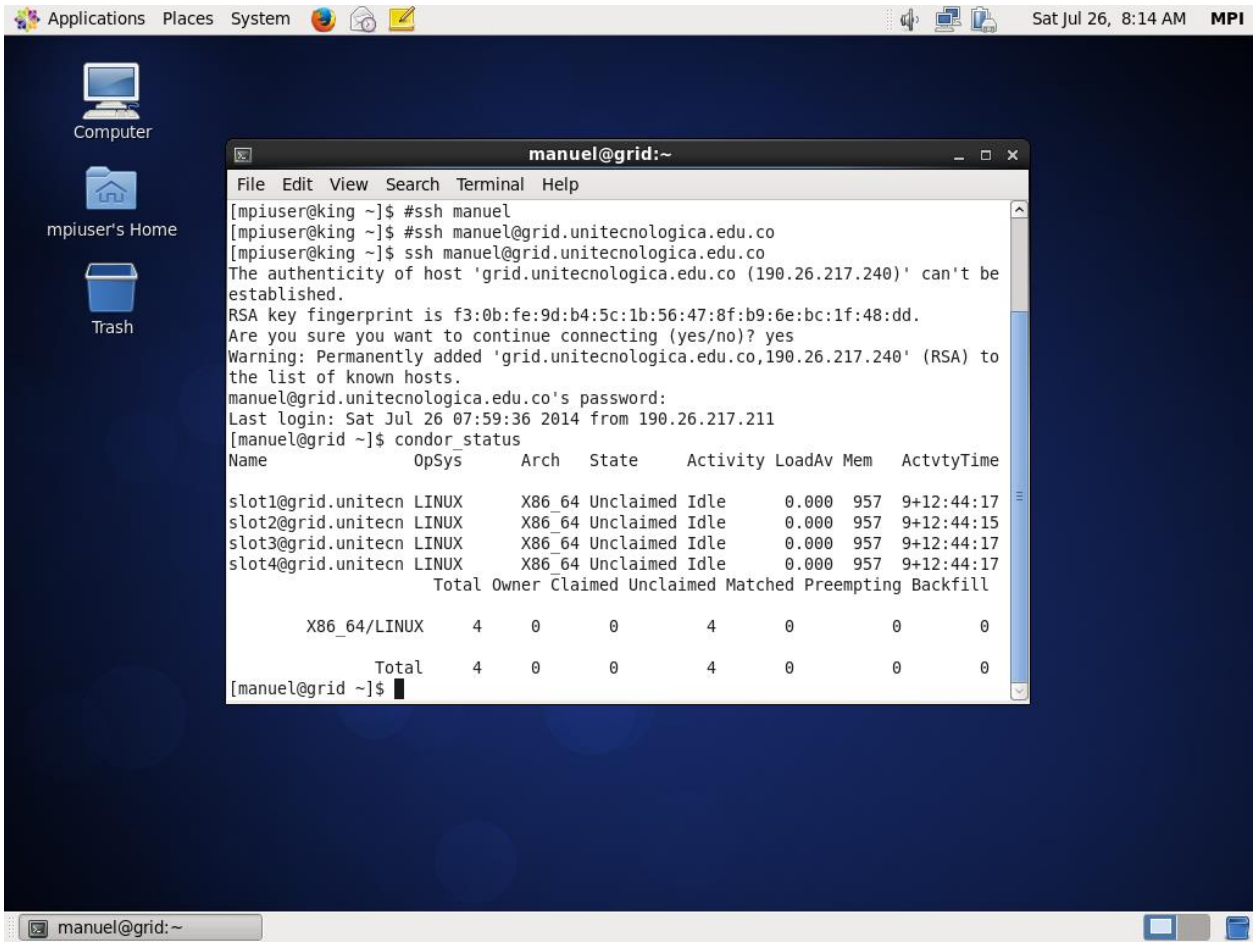


Figura 21: Procesadores disponibles en servidor grid HTCondor de UTB



A través de conexión SSH se accedió a servidor HTCondor de UTB. Figura 21 muestra acceso a servidor grid HTCondor de la Universidad Tecnológica de Bolívar. La Figura 21 exhibe que al utilizar comando de verificación del estado del cluster HTCondor, del que hace parte el servidor de UTB, se enlistan 4 nodos.

Figura 22: Configuración HTCondor en servidor grid de UTB

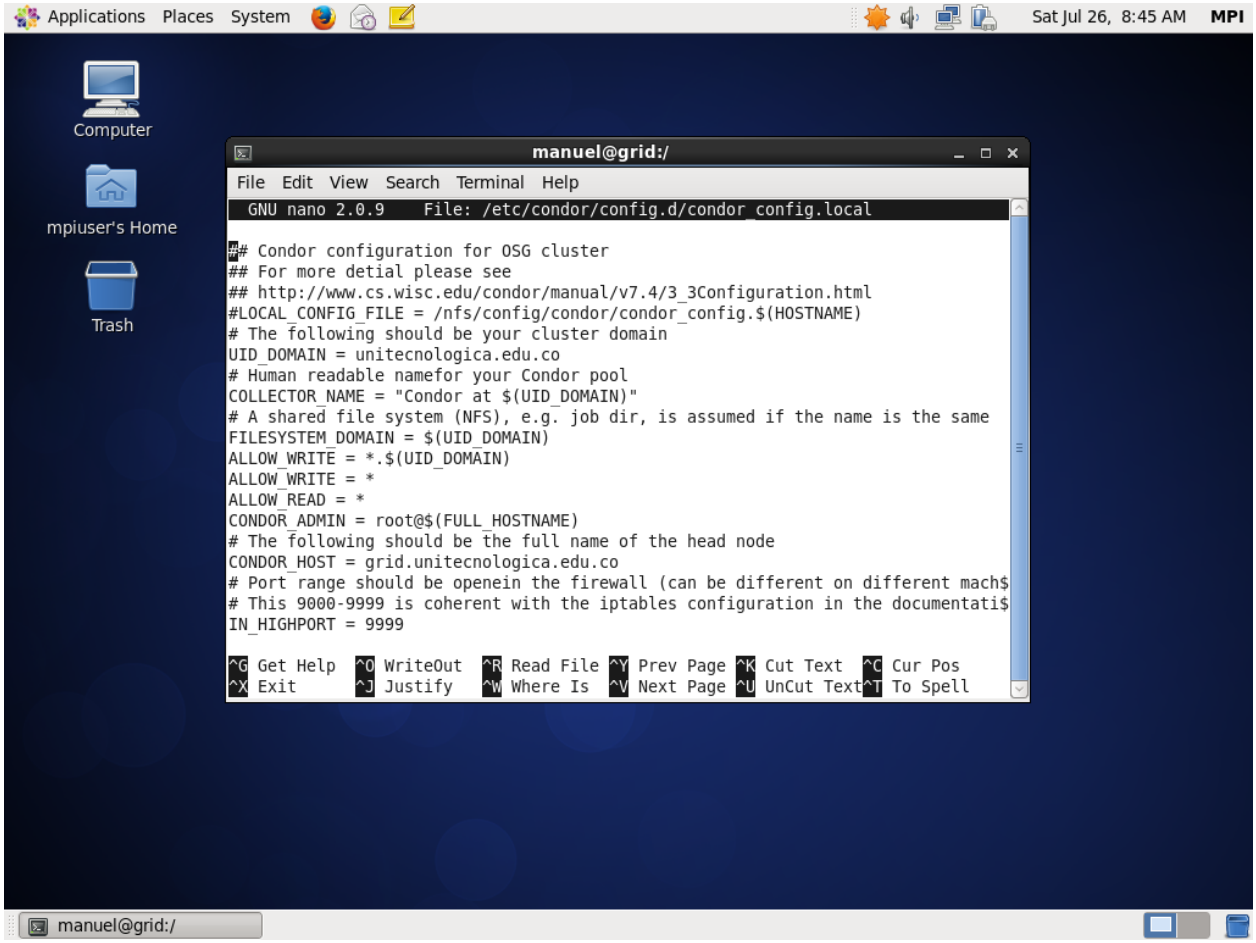
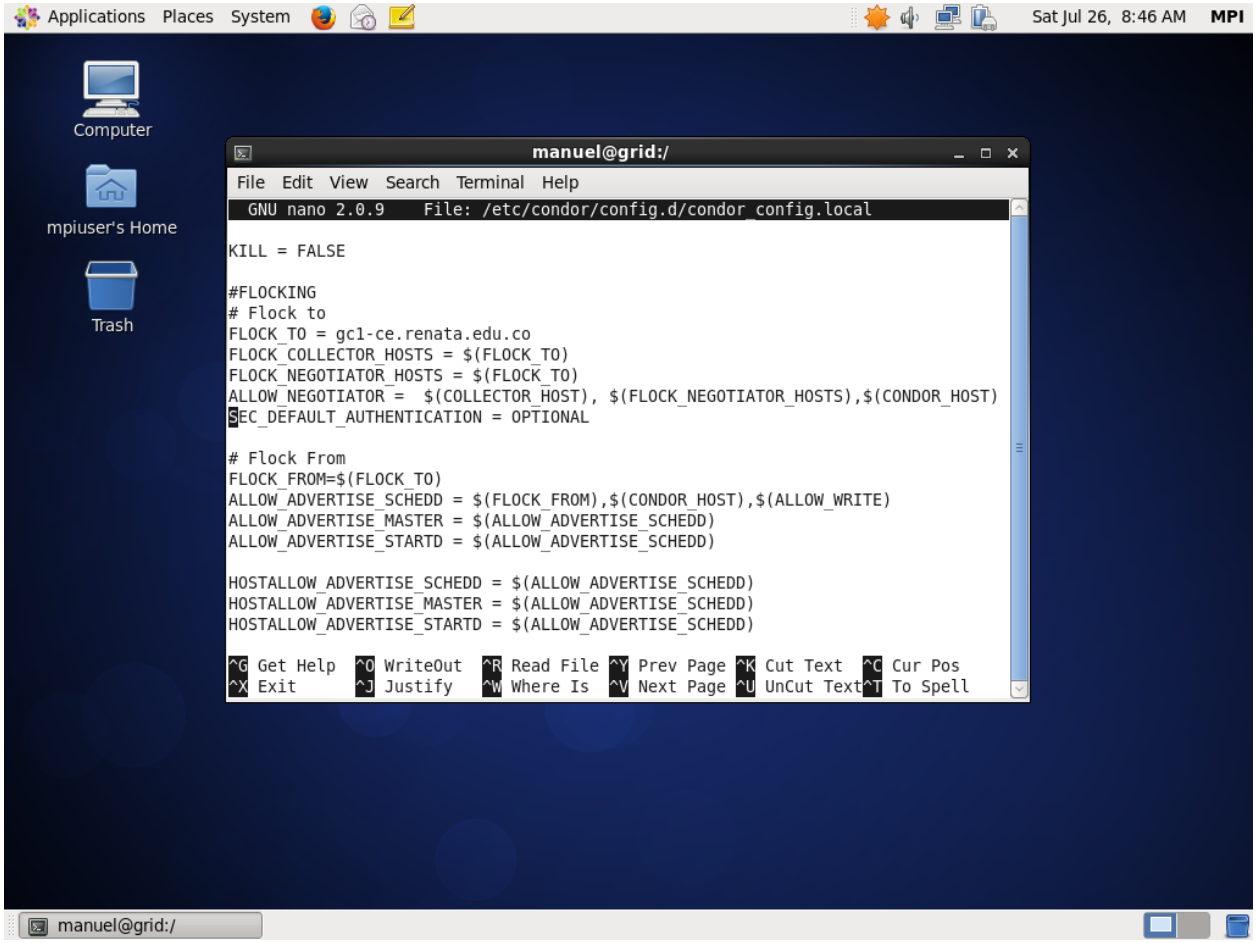


Figura 23: Configuración grid en HTCondor a través de Flocking



El archivo de configuración HTCondor del servidor grid de UTB, especifica como 'nodo maestro' dentro de cluster local al que pertenece, al Nodo Maestro, que es él mismo, ver Figura 22. Figura 23 muestra configuración Flock en el servidor, el cual permite que tareas que se envían en cluster local, al que pertenece servidor grid de la UTB, y no pueden ser procesadas por baja disponibilidad de dichos recursos computacionales se redireccione hacia otro cluster HTCondor. Un cluster de la red 3COA de RENATA está enrutado con el cluster de la UTB, lo cual permite que si se presenta la situación previamente descrita se conforme una grilla HPC, la que a su vez procesa las tareas enviadas, y donde el cluster remoto también actúa como ciber infraestructura HPC de respaldo del cluster HPC local.

Figura 24: Lista de nodos disponibles de cluster remoto de RENATA

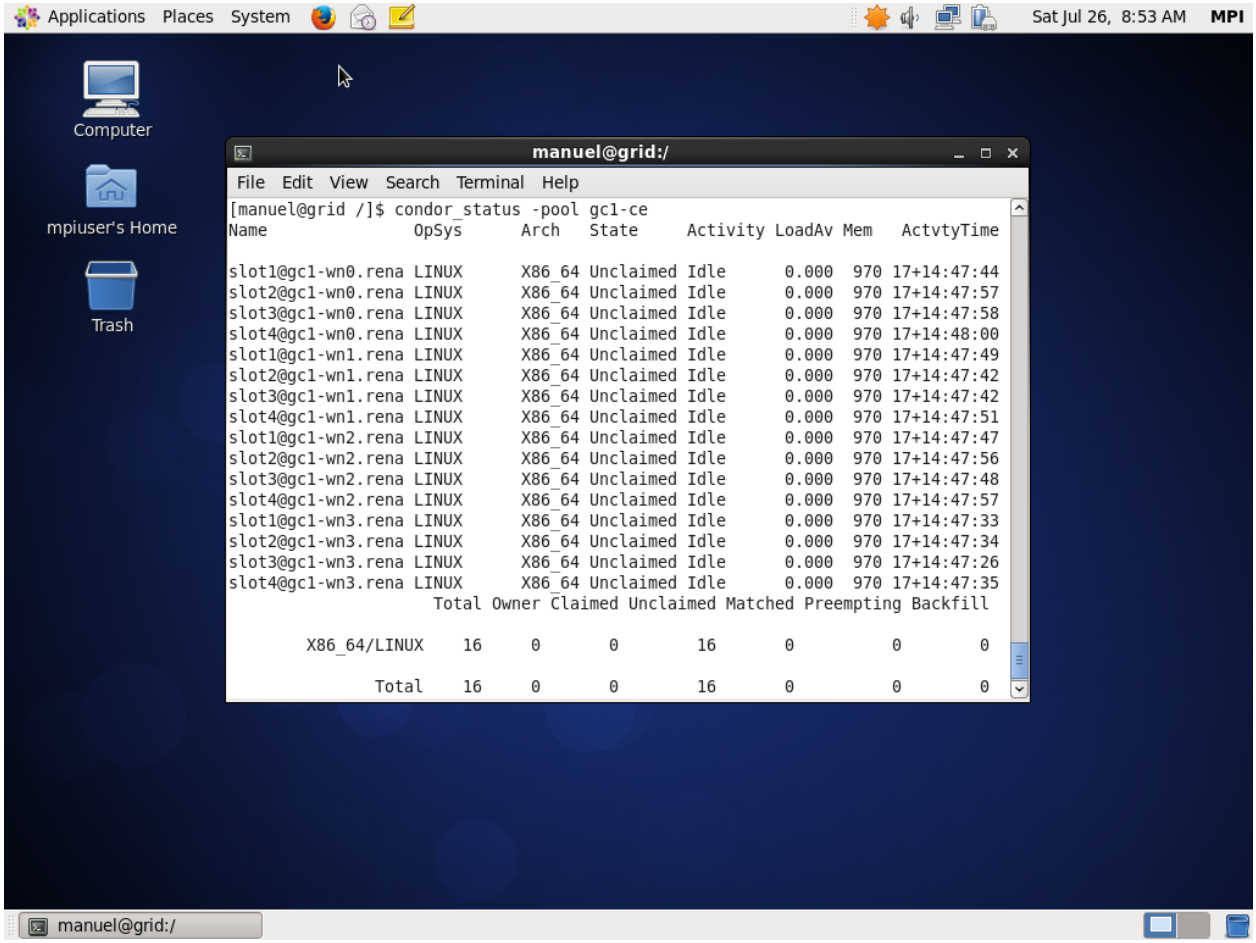


Figura 24 exhibe la salida de emplear comando HTCondor en servidor grid de UTB que enlista nodos remotos de cluster RENATA que hacen parte de la grid.

Para finalizar la presentación de resultados y análisis de las actividades del primer objetivo específico de este trabajo, se detalla a continuación, los procedimientos para instalar y configurar una herramienta de monitoreo de red, la cual proporciona medios para diagnosticar falencias de disponibilidad y desempeño de los recursos computacionales del ecosistema.

5.1.4. Configurar herramienta de monitoreo de red

La configuración de la herramienta de monitoreo de red Munin comprendió dos fases, similar a lo especificado en las dos actividades que preceden, es decir, se dividió la actividad en procedimientos que conciernen al Nodo Maestro del cluster y otras que conciernen a los Nodos Esclavos. En resumen, se requirió de la Instalación de web server y servicio en Nodo Maestro de cluster y también instalación de servicio cliente Munin en Nodos Esclavos.

5.1.4.1. Nodo Maestro

Para comenzar, en el Nodo Maestro se configuró repositorio 'Extra packages for Enterprise Linux' el cual proporciona paquetes software de alta calidad para distribuciones Enterprise Linux como lo son Scientific Linux, CentOS y Red Hat Enterprise Linux.

La herramienta de monitoreo de red Munin requirió de preinstalación y configuración de servidor web. Dicho servidor web se desplegó en Nodo maestro. Se inició dicho despliegue con instalación y configuración de servidor de aplicaciones web Apache, y motor de base de datos MySQL. Posterior a esto, se procedió con la configuración de MySQL, luego requirió iniciar su servicio, y finalmente se le configuró su respectivo usuario raíz.

El servidor web también se configuró para uso de páginas web PHP, como las utilizadas por Munin. Adicional a esto, se reinició servidor web apache, y luego a través del navegador web se accedió a página 'info.php', por medio de la cual se verificó si el lenguaje se instaló correctamente.

Se descargó e instaló la herramienta de monitoreo de red Munin en versión servidor y nodo. La versión servidor agrupa los diferentes nodos del cluster y despliega por servidor web el servicio de monitoreo. La versión nodo o cliente, permite registrar

nodo en servidor Munin para que el servicio web monitoree todos los nodos inscritos.

Se adicionó en archivo de configuración de servidor Munin la subred a monitorear. También, se reinició el servicio de servidor web para desplegar nuevo aplicativo web Munin. Dicho aplicativo se inició también en modo ‘node’, y adicionalmente, se configuró desde el Nodo Maestro, que hace veces de servidor Munin, las credenciales de acceso de usuario a través de la web. Finalmente, en el archivo de configuración de servidor Munin se especificó todos los nodos de la red a monitorear.

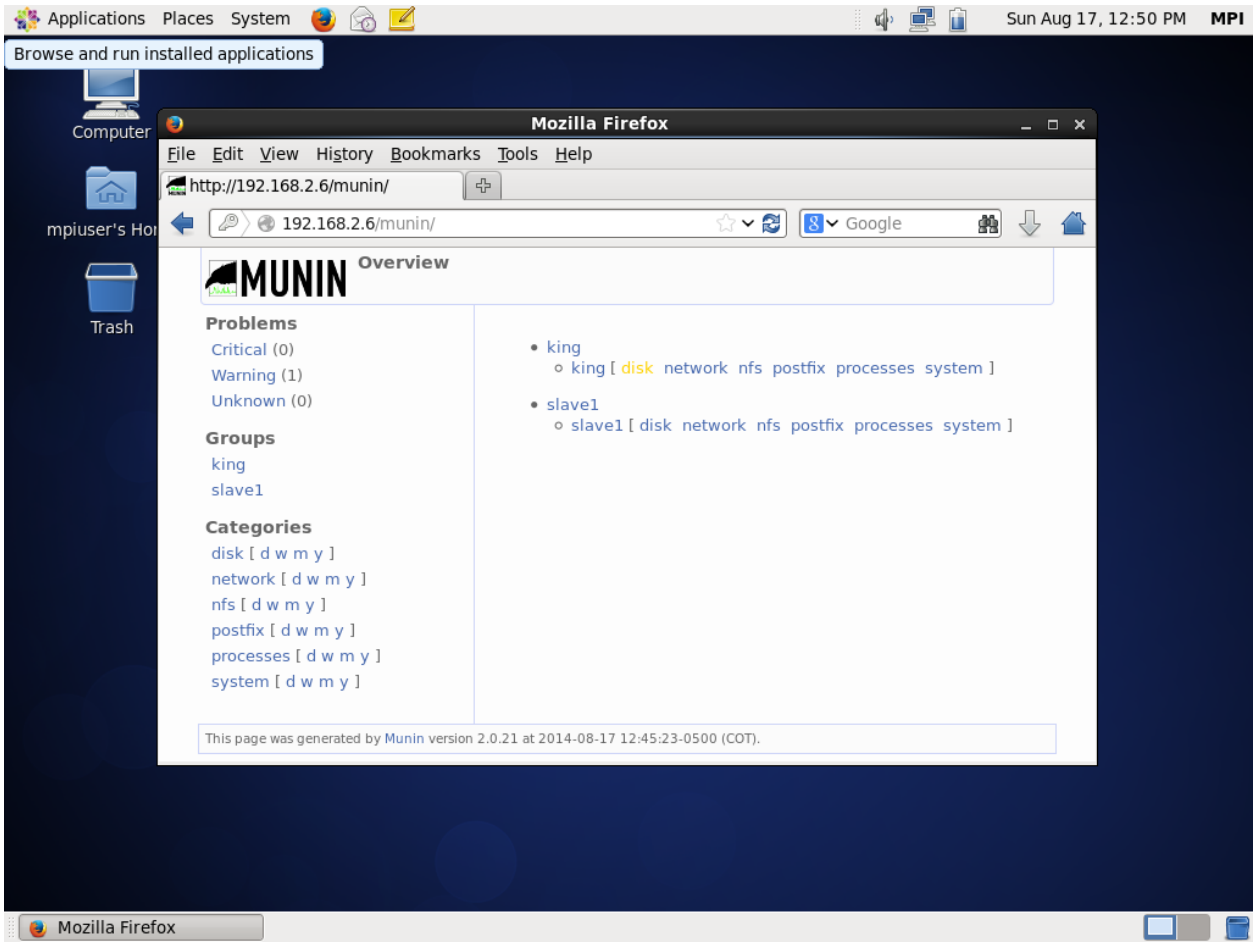
5.1.4.2 Nodo Esclavo

Para los Nodos Esclavos, se inició similar a como inició proceso en el Nodo Maestro. Se configuró repositorio ‘Extra packages for Enterprise Linux’, el cual proporciona paquetes software de alta calidad para distribuciones Enterprise Linux como lo son Scientific Linux, CentOS y Red Hat Enterprise Linux.

Luego, se descargó e instaló la herramienta de monitoreo de red Munin en versión nodo. La versión servidor instalada en el Nodo Maestro agrupa los diferentes nodos del cluster y despliega por servidor web el servicio de monitoreo. La versión ‘nodo’ o ‘cliente’ permite registrar los nodos en el servidor Munin, para que el servicio web monitoree todos los nodos inscritos.

Los archivos de configuración de cada Nodo Esclavo del cluster se les registró la dirección IP del nodo que hace veces de servidor Munin, en este caso Nodo Maestro.

Figura 25: Interfaz web de herramienta de monitoreo de red Munin

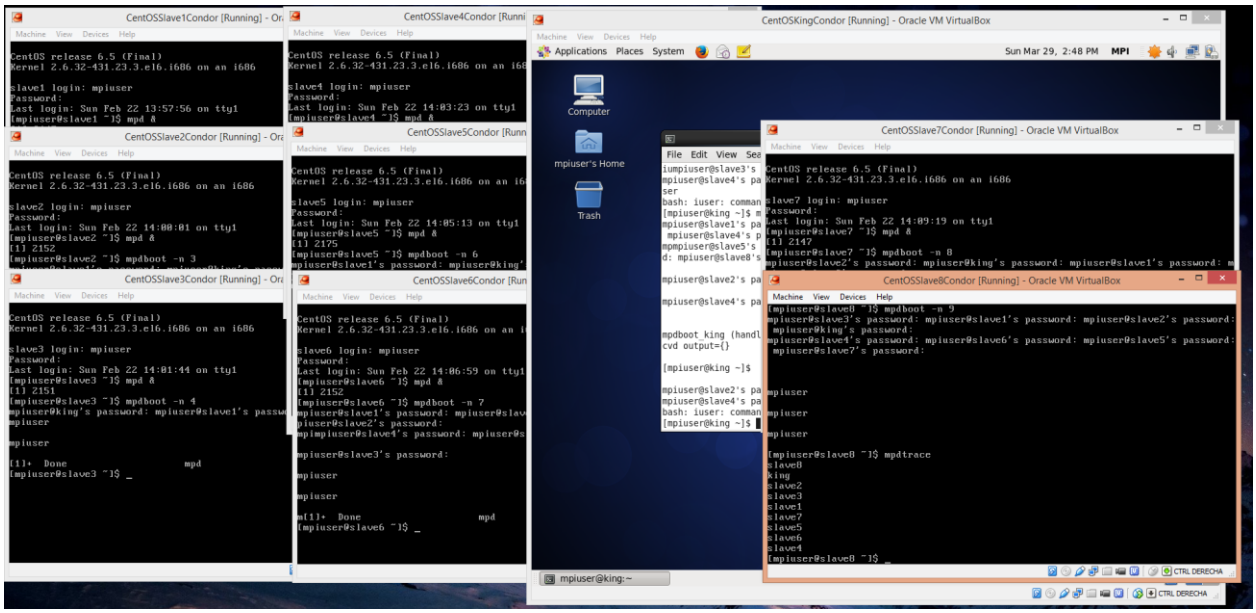


Finalmente, el ingreso a la interfaz web Munin expuso los nodos a monitorear en el cluster. Se puede verificar estado actual e histórico de valores de espacio en disco, procesamiento de las CPU, uso de memoría principal, throughput y otros. Figura 25 muestra interfaz web de Munin en la cual exhibe nodos de cluster en estado de monitoreo.

Al hacer recapitulación de actividades y resultados respecto a primer objetivo específico “Crear ambiente de desarrollo y ejecución a nivel hardware y software de algoritmos e-Science que requieren HPC en la UTB” se resume:

- Se estableció arquitectura software de sistema con base en requerimientos funcionales básicos y atributos de calidad. Se describió arquitectura propuesta de ecosistema de aprovechamiento en la que se expone requerimientos funcionales, atributos de calidad, aproximaciones arquitectónicas, vistas arquitectónicas de modelo de negocio, dominio, requerimientos, componentes, comportamiento, estático y despliegue.
- Se configuró cluster MPI con base en arquitectura de sistema. Dicho cluster se desplegó en Universidad Tecnológica de Bolívar y un entorno de pruebas para efectos de desarrollo de este trabajo; de ambos despliegues se extrajeron las figuras, ejecuciones y mediciones que emplea este escrito. Las especificaciones técnicas de los nodos que componen cluster MPI desplegado en UTB se rigen por los expresados en el modelo de despliegue de la arquitectura software del sistema, memoria RAM de 16 GB y 16 núcleos de procesamiento, mientras que la versión del cluster en entorno de pruebas cuenta con 4 núcleos de procesamiento y 8 GB de memoria RAM. Ambos clusters tienen configurados un Nodo Maestro y Nodos Esclavos. Figura 26 muestra 9 nodos del cluster de entorno de pruebas desplegado como hipervisor tipo 2.

Figura 26: Cluster de 9 nodos en entorno de pruebas



- La instalación y configuración del middleware HTCondor se realizó tanto en cluster MPI de UTB, como en cluster de entorno de pruebas. Cuando en este capítulo de presentación y análisis de resultados se habla de Cluster se habla de cluster MPI con middleware HTCondor incorporado. A diferencia del cluster de entorno de pruebas, el cluster de UTB tiene configurado el middleware HTCondor para enviar tareas hacia un cluster remoto de RENATA, trabajando dichas ciber infraestructuras en grid.
- La herramienta de monitoreo de red Munin se desplegó en cluster de pruebas y cluster de UTB.

Una vez se configuran los elementos hardware y software es posible desarrollar y ejecutar algoritmos escritos con sentencias MPI y código C/C++, además de utilizar las tradicionales ejecuciones y desarrollos seriales. Una gran parte de la ciber infraestructura es correctamente desplegada y es factible proceder a desarrollar y probar framework MPI inspirado en concepto de esqueletos algorítmicos.

5.2. PRODUCIR BASES DE UN FRAMEWORK A PARTIR DE PATRONES DE PROGRAMACIÓN PARALELA, DE MANERA QUE ALGORITMOS E-SCIENCE ANTERIORMENTE CODIFICADOS SECUENCIALMENTE O MEDIANTE HILOS/MPI SEAN ADAPTABLES AL MISMO

En las actividades de este segundo objetivo específico, se realiza multiples repasos de la literatura, consultas en comunidades académicas y en fuentes en línea con el objetivo de diseñar, implementar y probar pieza software inspirada en patrones de programación paralela como los esqueletos algorítmicos, con el fin de ofrecer alternativa y facilitar desarrollo de algoritmos e-Science que usan MPI, al abstraer detalles de implementación de dicha interfaz. A continuación, se expone detalles en referencia a actividades que buscan satisfacer la consecución del segundo objetivo específico de este trabajo.

5.2.1. Identificar decisiones de diseño a través de revisión de la literatura respecto a los esqueletos algorítmicos

El texto ‘Algorithmic Skeletons: Structured Management of Parallel Computation’ [32] y curso de ‘Programación Paralela con Esqueletos’ de Murray Cole [47] señalan gran número de decisiones/aproximaciones y argumentos respecto a diseño de esqueletos algorítmicos. A continuación se enlistan muchas de ellas, seguidas de decisión de implementadores de este trabajo, en referencia de si tomarlas o no para desarrollo de framework de paralelización propio.

Utilizar mayores niveles de abstracción en el diseño y escritura de algoritmos, en teoría, implica pérdidas en desempeño al momento de desplegar el mismo, a que si se utiliza atajos y explotaciones a bajos niveles de diseño y escritura del aplicativo. Si bien se compromete teóricamente el desempeño de los algoritmos, existen casos en que estos son tan complejos que lo que inclusive es más difícil, es escribir estos a bajo nivel, lo que abre campo para ofrecer alternativas de solución al subir niveles de abstracción [32] [47].

Se define tres niveles de abstracción: Alto, medio y bajo. En el nivel alto el usuario no lidia con paralelización, los modelos computacionales se basan en modelos funcionales, lógicos, o de flujo de datos; utiliza sentencias declarativas que define propiedades de una solución en vez de secuencia de operaciones; no existe noción de control sobre el flujo de trabajo. En el nivel medio el usuario escribe código paralelo y se encarga de descomponer la solución en colección de procesos concurrentes; la propia abstracción lidia con mecanismos de interacción entre procesos. En nivel bajo de abstracción, usuario lidia con paralelización explícita y distribución de los procesos de acuerdo a la topología de red [32] [47]. En este trabajo, se estableció nivel medio de abstracción como objetivo de desarrollo, con el fin de dar noción al usuario sobre labor de paralelización, pero con noción de la descomposición y ocultamiento de tareas como la distribución, reparto de datos y código, comunicación y sincronización.

De acuerdo a Cole, en sistemas altamente abstractos existe tara (overhead) en mayor proporción que a un sistema explícitamente paralelo, ya que en estos últimos se puede mapear solución del problema respecto a la topología de red, mientras que en sistemas de más abstracción el usuario final generalmente no tiene acceso a estos detalles de implementación [32].

La aparición de técnicas que resuelven determinado tipos de problemas e-Science permite la designación de una de estas técnicas a un esqueleto algorítmico [32]; en este trabajo se descartó uso de una estructura universal, ya que esto limita implementaciones paralelas eficientes. Cole También, se plantea creación de múltiples esqueletos algorítmicos y la combinación de estos para crear otros [32]. En este trabajo también se determinó crear una estructura inicial inspirada en una de estas estrategias o técnicas, pero no se aplica concepto de esqueleto algorítmico en sentido literal propuesto por Cole, con el objetivo de dar más flexibilidad y variedad en soluciones que usuario puede hacer uso. Aun así, inclusive el mismo Murray Cole enmarca soluciones híbridas o inspiradas en su concepto dentro de la familia de concepto de esqueletos algorítmicos [47].

Cole propone como medio de implementación de esqueletos algorítmicos, el uso del concepto de funciones de orden superior [32] [47]. Para este trabajo, se declinó

uso inicial de este concepto, ya que por efectos de alcance, restricciones de tiempo inherentes, se planteó una aproximación simplificada que buscara maximizar garantía de funcionalidad, y que de lograrse se pudiera refactorizar con concepto de funciones de orden superior y apuntar a que framework proporcionara funcionalidad bajo este concepto. Por otra parte, el que el framework no utilice directamente concepto que propone el autor, no exime que el software no sea clasificado dentro de la familia de esqueletos algorítmicos [47].

El paradigma de programación que se destinó para desarrollar framework de paralelización de este trabajo, es un híbrido entre programación procedimental y orientado a objetos. Para efectos de satisfacción de atributos de calidad como: mantenibilidad, portabilidad y usabilidad, se planteó desarrollar framework utilizando solamente paradigma de programación orientada a objetos. Si bien el ideal es utilizar este paradigma orientado a objetos, por razones de aprendizaje en uso de paradigma de programación distribuida con MPI por parte de los implementadores de este trabajo de investigación, y que se buscaba garantizar funcionalidad en menor tiempo posible de acuerdo a ajustes en cronograma de entrega de este trabajo, se buscó desarrollar el software con base en múltiples instructivos, que en su mayoría enseñan MPI mediante paradigma de programación procedimental. Si efectivamente el software funcionaba utilizando paradigma procedimental, se incorporaría refactorización con paradigma orientado a objetos en la medida que se garantizara funcionalidad de la pieza software.

Cole afirma que el desempeño de las implementaciones tiene injerencia directa de relación memoria-comunicación-procesadores. Recomienda distribuir los procesos de acuerdo a vecindad entre nodos/procesadores. Una estructura práctica para estos efectos es un arreglo de dos dimensiones, una matriz cuadrada para ser más precisos [32]. Para la implementación del framework software de este trabajo, se planteó la creación adicional de archivo de hosts, el cual permitiera que MPI asigne automáticamente un rango/ID de manera secuencial a cada nodo del archivo, con lo cual se pudiera distribuir procesos a procesadores de forma eficaz mediante dicho archivo y estructuras de datos tipo matriz, explícitas dentro del framework.

Murray Cole propone 4 esqueletos principales, cada uno con múltiples variantes, los cuales a su vez abordan diferentes tipos de problemas científicos, como lo es resolución de transformada de fourier discreta, multiplicación de matrices, integrales aproximadas, construcción de árboles recubridores mínimos, arreglos lógicos programables, descomposición de matrices LU, sistemas matriz banda de ecuaciones lineales, caminos cortos en un grafo, y otros [32] [47]. En el framework de este trabajo se buscó inicialmente desarrollar estructuras inspiradas en el esqueleto 'Divide & Conquer' y sus variantes. Teniendo en cuenta el tiempo que tomara desarrollar una variante del esqueleto y de acuerdo al cronograma ajustado de entrega de este trabajo, se tomría decisión de si se procedía a desarrollar otras variantes o no.

Se comenzó el desarrollo del framework con base en esqueleto 'Divide & Conquer', variante 'FIxed Degree, k=2'. La variante binaria de esqueleto 'divide y vencerás' plantea uso de matriz de procesadores para distribuir e intercomunicar procesadores acorde con estrategia de subdivisión de tareas [32], de manera que problemas e-Science, tales como aquellos de reducción de datos, puedan ser resueltos satisfactoriamente. Por lo tanto, se requería desarrollar dichas operaciones dentro del framework, las cuales debían ser transparentes para el usuario final, con excepción de operación de descomposición, del cual éste tendría noción en alto nivel de abstracción.

El esqueleto 'Fixed Degree Divide & Conquer,k=2' plantea uso de matriz lógica de procesadores para distribuir procesos descompuestos por el mismo esqueleto. Cole plantea mapear un 'árbol binario completo' dentro de la matriz, con el objetivo de que un nodo/procesador del cluster tenga enlaces con un determinado nodo hijo/hoja y otro nodo padre; dicho árbol tiene el nodo raíz que se ubica en la coordenada céntrica de la matriz. Con base en fórmula algebraica se determina los enlaces entre los nodos del cluster. Se toma cada nodo activo o vértice del árbol con coordenadas cartesianas (a, b), y dependiendo del si el valor total de vértices 'v' y la profundidad 'd' del vértice en cuestión son pares, impares o mezcla de ambos, se determinan los nodos hojas de dicho vértice o nodo. Las Figuras 27 y 28, exponen las fórmulas que permiten lo anteriormente descrito.

Figura 27: Fórmulas algebraicas para determinar enlaces binarios entre nodos del cluster. Figura tomada de texto 'Algorithmic Skeletons: Structured Management of Parallel Computation' de Murray Cole

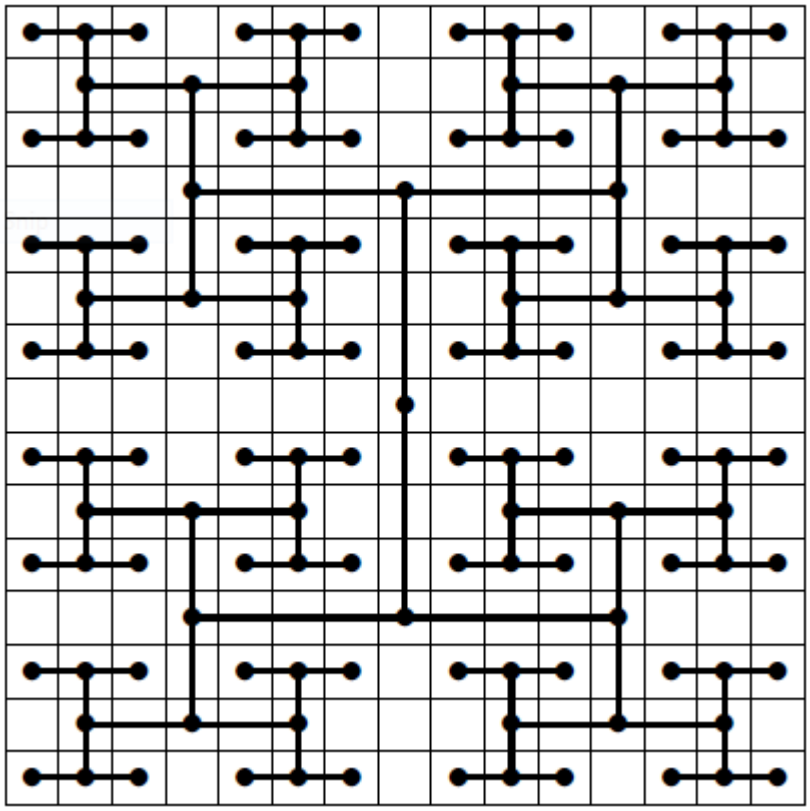
$$\left(a, b + 2^{\frac{1}{2}\left(\log_2\left(\frac{v+1}{2}\right)-d-2\right)}\right) \text{ and } \left(a, b - 2^{\frac{1}{2}\left(\log_2\left(\frac{v+1}{2}\right)-d-2\right)}\right)$$

if $\log_2\left(\frac{v+1}{2}\right)$ and d are both even or both odd, or

$$\left(a + 2^{\frac{1}{2}\left(\log_2\left(\frac{v+1}{2}\right)-d-1\right)}, b\right) \text{ and } \left(a - 2^{\frac{1}{2}\left(\log_2\left(\frac{v+1}{2}\right)-d-1\right)}, b\right)$$

if one of $\log_2\left(\frac{v+1}{2}\right)$ and d is even and the other odd,

Figura 28: Construcción de árbol binario completo en ‘H’. Figura tomada de texto ‘Algorithmic Skeletons: Structured Management of Parallel Computation’ de Murray Cole



Estas operaciones a nivel del framework software de este trabajo, se materializó a través de matrices y vectores que tomaron papel de estructuras que manejan aspectos como coordenadas cartesianas, profundidades del árbol e identificación de los mismos nodos. La matriz se dimensionó de 3 x 3, para un total de 9 nodos, (se utilizó cluster de pruebas de 9 nodos, mencionado al final del capítulo de resultados y análisis del primer objetivo específico), y se fijaron estos valores porque Cole propone matrices cuyas dimensiones permitan mapeo de árbol binario en ‘H’ o ‘completo’ [32], con lo cual como dimensiones mínimas para realizar pruebas razonables de funcionalidad del framework es de 3 x 3.

El proceso de mapeo del árbol en la matriz en esta tesis, implicó determinar el número de procesadores/nodos a utilizar, es decir p = 3 * 3 = 9, el número de vértices dentro de la matriz de procesadores que realiza trabajo, v = 7, también se calculó las profundidades/nivel del árbol, las cuales se establecieron entre 0 a 1, acorde con el límite, 0 ≤ d ≤ Log2(v+1/2) – 1. Estos valores, de la mano de fórmulas expuestas en Figura 27, permitieron el cálculo de las coordenadas cartesianas de los nodos hijos de un procesador cualquiera de la matriz; inicialmente, se empezaron los cálculos empleando el nodo raíz/central de la matriz, el cual se ubicó en las coordenadas cartesianas (0,0). Una vez se calcularon los nodos hijos del procesador ráiz/central de la matríz, se prcedío a realizar los mismos cálculos para obtener los hijos de estos hijos, por medio de las coordenadas cartesianas ya obtenidas. La Figura 29 expone los valores iniciales con los que se procedió a mapear el árbol binario. La Figura 30 muestra las coordenadas cartesianas dentro de la matriz, para cada nodo/procesador, así como la de sus hijos. Finalmente, la Figura 31 muestra un plano cartesiano, empleado como guía para ejecutar el mapeo, mientras que la matriz adyacente muestra con un esquema de colores la jerarquía del árbol binario mapeada en la matriz; el nodo central/raíz se identifica con color azul, sus nodos hijos se identifican con un tono rojo, mientras que los respectivos nodos hijos de estos se distinguen con un tono amarillo.

Figura 29: Parámetros iniciales para mapear árbol binario en matriz de procesadores

p|n	v	d_i	log_2(v+1/2)
3*3	7	0	2
		1	
		2	

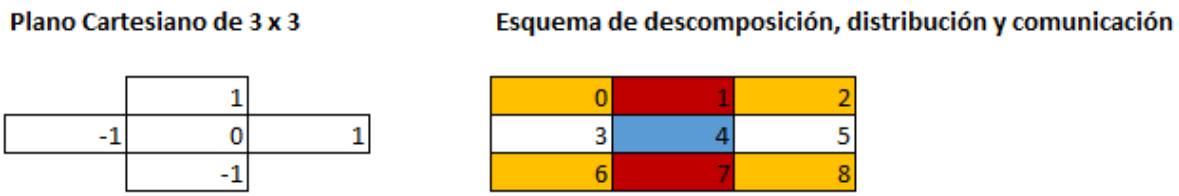
Figura 30: Cálculo de coordenadas cartesianas que permiten mapear árbol binario en matriz de procesadores. El cálculo se realiza empleando las fórmulas expuestas en la Figura 27 y los valores iniciales de la Figura 29

	d=0	
para vertex (0,0)	**X**	**Y**
hijo 1:	0	1
hijo2:	0	-1

	d=1	
para vertex (0,1)	**X**	**Y**
hijo1:	1	1
hijo2:	-1	1

	d=1	
para vertex (0,-1)	**X**	**Y**
hijo1:	1	-1
hijo2:	-1	-1

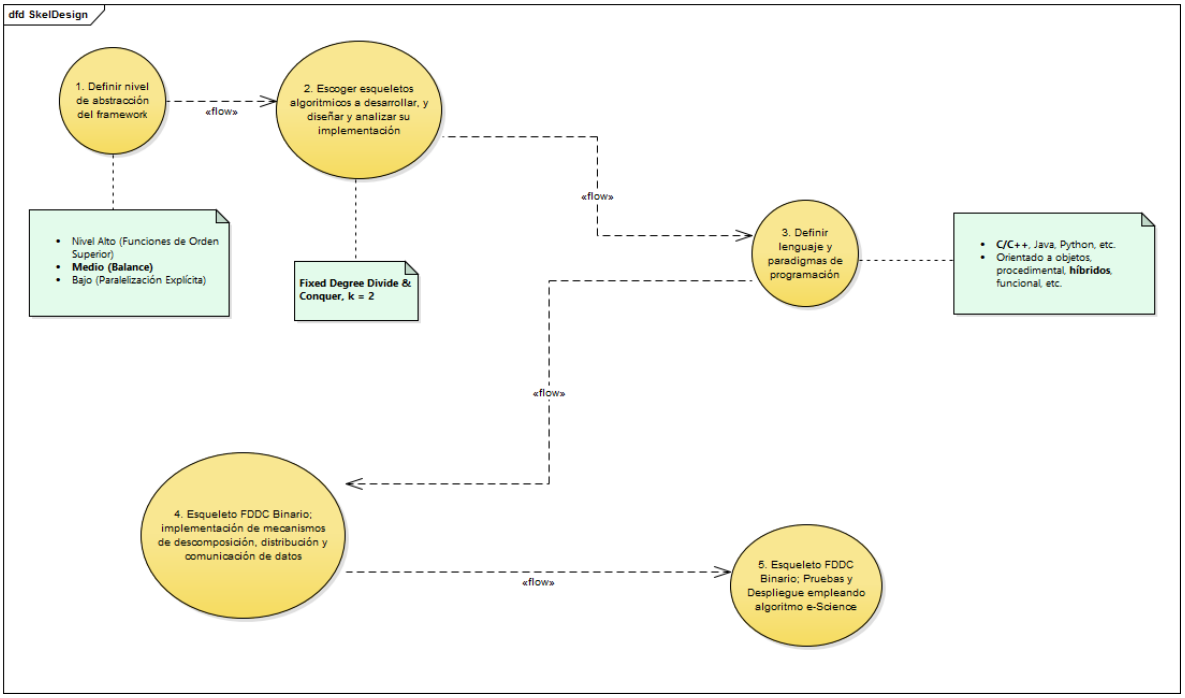
Figura 31: Plano cartesiano guía y matriz de procesadores final



Por otro lado, se escogió lenguaje de programación C++, (en ocasiones mezclado con lenguaje C), como medio de desarrollo de framework por varias razones. Es un lenguaje basado en C, permite implementación de paradigma de programación orientado a objetos, programación funcional por medio de plantillas, y otros paradigmas modernos, perfecto para satisfacer algunos atributos de calidad del sistema software. Es compilable con lenguaje C y sentencias MPI. Es un lenguaje muy utilizado por la comunidad científica en conjunto con C, FORTRAN, Python y otras librerias y lenguajes cientificos modernos, pero destaca por: Es más amigable de usar que C y ofrece mayores funcionalidades como programación orientada a objetos y uso de estructuras como plantillas. Es más amigable de usar que

FORTRAN. Es debatible que desempeñe mejor que C y FORTRAN, pero no con relación a Python y otros lenguajes como Matlab que son escritos sobre lenguaje C. Existe gran cantidad de documentación de referencia, librerías y comunidades científicas en línea que facilita aprovechamiento de lenguaje C++ y/o C. La Figura 32, a continuación, muestra en alto nivel el proceso de diseño del framework.

Figura 32: Esquema de diseño de Framework de paralelización inspirado en esqueletos algorítmicos



Con las consideraciones de diseño del framework establecidas, prosigue establecer mismas decisiones pero esta vez respecto a cómo el diseño del framework, en sí, aspectos de descomposición, distribución, comunicación y sincronización, se traducen a nivel de implementación utilizando interfaz MPI. En siguiente ítem del texto se reporta consideraciones de diseño que traducen diseño de framework a código fuente utilizando MPI.

5.2.2. Identificar decisiones de diseño a través de revisión de conceptos sobre MPI

Con base en Texto ‘Algorithmic Skeletons: Structured Management of Parallel Computation’ [32] de Murray Cole, se identifica decisiones de diseño respecto a que se debe hacer para plasmar directrices de cómo crear determinado concepto de esqueleto algorítmico en código fuente. Adicionalmente, dicha acción implica traducir tales directrices en código fuente con sentencias MPI. El texto ‘MPI - The Complete Reference’ de Wopp, et al [35] y recurso en línea ‘Message Passing Interface (MPI)’ de la Universidad de Livermore [48] proporcionan información que ayuda a determinar qué operación u operaciones MPI permiten correcta implementación de los conceptos de esqueletos algorítmicos propuestos por Cole, en este caso para crear pieza software inspirada en estos esqueletos.

La descomposición, distribución, comunicación y sincronización de procesos en nodos del cluster mediante una matriz lógica de árbol binario ‘H’ a nivel MPI, se presumía podia abordarse con uso de conceptos y rutinas de ‘topología virtual’, fuere por ‘coordenadas cartesianas’ o por ‘grafos’. Las topologías virtuales por coordenadas cartesianas permitían mapeo de nodos padre-hijos, en cual cada nodo se le asocia un nodo destino a donde envía datos, y otro nodo fuente de donde recibe datos; con referencia al framework de este trabajo, en el caso de los nodos hojas lo anterior no es congruente, ya que las hojas deben recibir de, y enviar al mismo nodo; la operación shift dentro de las topologías cartesianas no brinda esta flexibilidad debido a parámetros fijos de desplazamiento y dirección del desplazamiento, por ello esta topología no aplicó, pues no se puede implementar tan siquiera tareas de descomposición y distribución de la forma deseada. Por otra parte, la topología por grafos es viable en cuanto a tareas de descomposición y distribución, ya que se puede especificar con mayor flexibilidad por cada nodo, cuál o cuáles son sus vecinos, y así mapear la estructura padre-hoja que requiere el esqueleto ‘Fixed Degree Divide & Conquer, k=2’, pero para utilizar operaciones de comunicación colectivas o ‘punto a punto’, se requiere utilización de otras propiedades adicionales de MPI; y es que con sólo utilizar estas propiedades sin necesidad de emplear las topologías virtuales, se garantiza manejo de tareas de descomposición, distribución, comunicación y sincronización en el framework.

Las propiedades y operaciones MPI que permiten el manejo de tareas de descomposición, distribución, comunicación y sincronización en el framework son: manejo de comunicadores MPI, manejo de intercomunicadores MPI, operaciones de comunicación colectiva con bloqueo como scatter y gather, y operaciones de comunicación punto a punto con bloqueo como send y receive. Los comunicadores MPI se utilizan para enlazar y jerarquizar trios de nodos, es decir nodo padre con sus dos hojas/hijos, mientras que los intercomunicadores se emplean para enlazar cada comunicador a través de los nodos padres de cada uno de estos. Se utiliza operaciones de comunicación con bloqueo para garantizar que comunicación de datos sean procesadas por el programa en cada nodo del cluster. Las operaciones de comunicaciones colectivas se utilizan para que los nodos padre de cada comunicador envie conjuntos de datos a cada hoja/hijo, estos los reduzcan y reenvien resultado a nodo padre, mientras que las operaciones de comunicación punto a punto, se emplean para enviar un dato entre los nodos padres de los comunicadores, esto mediante un intercomunicador.

En el reporte de la actividad a continuación, se presenta decisiones de diseño recolectadas en las primeras dos actividades de este objetivo específico, plasmadas a nivel de código fuente.

5.2.3. Programar framework en lenguaje C/C++ con tecnología MPI

El framework de paralelización inspirado en esqueleto ‘Fixed Degree Divide & Conquer, k=2’ consta de una clase llamada ‘FDDC’, que a su vez contiene un método denominado ‘FDDCK2’. Este método contiene lógica que descompone, distribuye, comunica y sincroniza los procesos con los procesadores del cluster. Todas estas tareas se desarrollaron y ejecutaron a través de la manipulación de estructuras de datos como arreglos uni y multidimensionales, y estructuras y conceptos MPI descritos en las dos actividades iniciales de este objetivo específico. El método FDDCK2 también contiene en su código fuente el llamado de las operaciones o métodos de otra clase, denominada ‘F’. El framework en el estado de desarrollo para esta tesis, logra resolver algoritmos e-Science abordados por reducción de datos.

La clase F contiene operaciones en las que usuario del framework escribe su código. El usuario debe entender secuencia de llamado de dichas operaciones para poder ejecutar su algoritmo de la manera deseada. Debido a que el usuario debe entender la secuencia de ejecución de las operaciones de esta clase, este tiene conocimiento de la tarea de descomposición del framework, pero esto a nivel muy abstracto; al usuario se le daría instrucción de ello mediante una breve explicación dentro de la misma clase. El usuario final del sistema no se vería en la obligación de modificar el contenido de la clase FDDC, por tanto no se ve en obligación de entender detalles de implementación de la clase que agrupa lógica y detalles técnicos principales de esta pieza del framework.

Finalmente, mientras que el usuario escribe su código en el cuerpo de las operaciones de la clase F, este debe designar la clase 'main' como clase ejecutable al momento de compilar su programa. Así como sucede con clase FDDC, el usuario no se ve en la necesidad de modificar contenido de la clase main.

Figura 33: Diagrama de clases de framework software inspirado en esqueleto 'Fixed Degree Divide & Conquer, k=2'

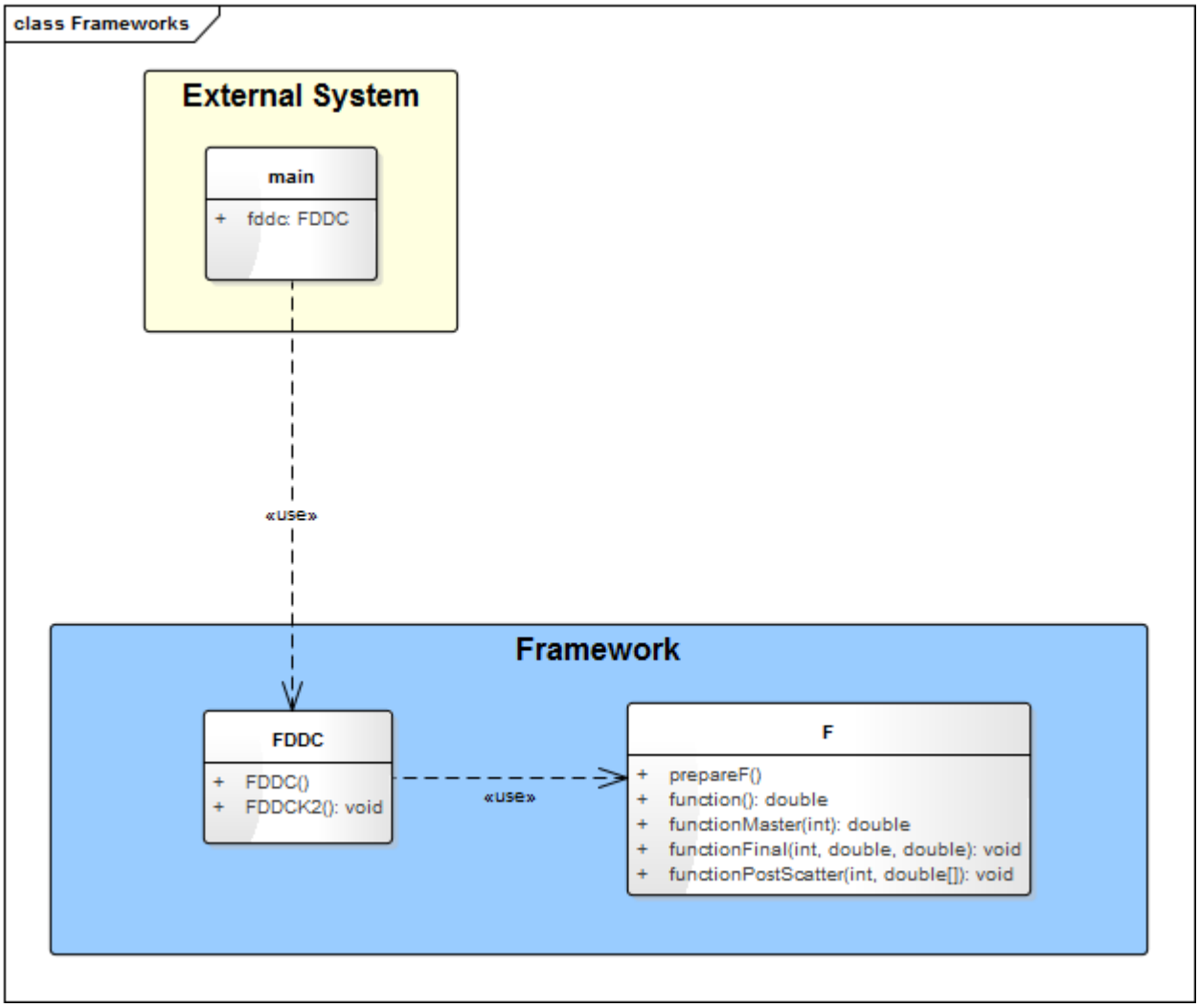


Figura 33 muestra diagrama de clase de software framework para desarrollar algoritmos e-Science, que abstrae detalles de implementación de MPI. Los anexos 02, 03 y 04, exhiben en detalle el contenido de las clases, archivos fuentes y cabeceros ‘FDDC.cpp’, ‘F.h’ y ‘main.cpp’.

En el código fuente que se escribió dentro del archivo cabecero ‘F.h’ del framework, se escribió dentro de cinco métodos. El código es secuencial dentro de los métodos del archivo cbecero, y se ejecutan en una determinada secuencia acorde a cuando lo llama el archivo fuente ‘FDDC.cpp. El método ‘prepareF()’ es aquél donde el usuario puede inicializar atributos o propiedades del algoritmo. Este método

inicializa el arreglo ‘inputArray[]’, el cual es utilizado para almacenar valores de iteraciones para las reducciones de datos que realizan algunos nodos del cluster. La función ‘function()’ es aquella que realiza las computaciones más básicas, realiza cálculos como deseé el usuario. Esta función la llaman los nodos hojas del árbol binario. El método ‘functionPostScatter()’ está diseñada para computar y realizar reducción de datos, esto a nivel de los nodos padres del árbol, a excepción del nodo raíz. Igualmente, allí se puede personalizar el código a gusto del usuario. El método ‘functionMaster()’ realiza aproximación de los valores devuelto por la reducción de la función ‘functionPostScatter()’. Finalmente, el método ‘functionFinal()’ realiza la aproximación final de las reducciones de datos realizadas por los nodos hojas del nodo ráiz.

En el último objetivo específico a continuación, se detalla resultados y análisis respecto a confrontación de complejidad de desarrollar un algoritmo e-Science utilizando framework inspirado en esqueletos, y desarrollar el mismo algoritmo utilizando MPI directamente.

5.3. EVALUAR A TRAVÉS DE MÉTRICAS SOFTWARE DE MCCABE, HALSTEAD Y OMAN/HAGEMEISTER, COMPLEJIDAD DE PROGRAMAR ALGORITMO E-SCIENCE UTILIZANDO FRAMEWORK VERSUS MPI PURO EN ECOSISTEMA DE APROVECHAMIENTO

Para el último objetivo específico de este trabajo, se busca realizar comparativo en referencia a que tan complejo/facil se torna para un usuario, desarrollar un algoritmo e-Science utilizando framework inspirado en concepto de esqueletos algorítmicos, versus utilizar MPI puro. A continuación, se detalla resultado de ejecución de actividades pertinentes a programación de ambos tipos de programas utilizando como problema e-Science el cálculo aproximado del número Pi, posterior a esto, se calcula múltiples métricas de complejidad, como las Halstead, la Ciclomática, Índice de Mantenibilidad, junto a comparativos de tiempo de ejecución. Finalmente, se realiza análisis referente a la obtención de estas métricas.

5.3.1. Programar y probar algoritmo e-Science en MPI puro y ejecutar en ciber infraestructura cluster MPI

La versión MPI pura del cálculo aproximado del número Pi, se desarrolló de forma simplificada. Para la versión MPI pura, a diferencia de las sentencias MPI utilizadas para desarrollar framework, no se utilizó sino un solo comunicador MPI, y un solo tipo de operación de comunicación colectiva, la de tipo reducción. Por demás, se utilizaron propiedades y sentencias que cualquier programa MPI utiliza típicamente, estas son, inicialización de entorno MPI, registro de los nodos/procesadores del cluster, y finalización de entorno MPI.

En cuanto al cálculo aproximado del número Pi, se utilizó paradigma de programación orientado a objetos para representar dicha operación. Se empleó las clases 'GeometricalManager', 'Rectangle' y 'Circle' para aproximar el número Pi con base en cálculo de área de un cuadrado de lado 'r', dividido entre área de un circulo de radio 'r', todo multiplicado por 4. En el algoritmo MPI puro, se aproximó el valor de Pi miles de veces y al final se promedió el valor. Para revisar implementación del cálculo aproximado del número Pi con MPI puro, revisar anexo 05.

Debido a que proceso de desarrollo de cualquier pieza software implica iteración constante entre pruebas, programación y ejecución, en este trabajo se evidencia como acción de pruebas del algoritmo la eficaz ejecución de este. Se realizaron 10 ejecuciones del algoritmo en un cluster MPI de la UTB, el cual consta de 16 núcleos de procesamiento y 16 GB de memoria principal; se utilizaron 9 procesadores del cluster.

Para las 10 ejecuciones mencionadas; se midió el tiempo de ejecución en términos de tiempo de 'usuario' y de 'sistema'. El tiempo de ejecución de usuario es el tiempo que tarda el procesador en ejecutar el código de una tarea de modo usuario, es decir, este tiempo no contabiliza interrupciones por parte del kernel y bloqueos de otras tareas externas. El tiempo de sistema, es el tiempo requerido para procesar operaciones de la tarea que requiere de intervención del kernel, como lo son por ejemplo, reservación de memoria, lectura/escritura de archivos, entre otros. El tiempo de ejecución denominado 'real', se refiere al tiempo de completo del llamado

a la tarea, es decir, puede incluir tiempo requerido en operaciones de entrada/salida del usuario, o interrupción por procesamiento de tareas externas.

Para calcular el tiempo total de ejecución o procesamiento del algoritmo, se sumaron los tiempos de ejecución de 'usuario' y 'sistema'. En Tabla 1 se aprecia resultados de las ejecuciones del algoritmo de cálculo aproximado del número Pi utilizando MPI puro.

Tabla 1: Tiempos de ejecución de cálculo aproximado de numero Pi utilizando MPI puro

	Tiempo Total de CPU	**Tiempo Usr (s)**	**Tiempo Sys (s)**	**Pi**
	1.491	1.287	0.204	3.1416
	1.397	1.275	0.122	3.1416
	1.479	1.368	0.111	3.1416
	1.458	1.319	0.139	3.1416
	1.591	1.387	0.204	3.1416
	1.603	1.409	0.194	3.1416
	1.506	1.341	0.165	3.1416
	1.556	1.368	0.188	3.1416
	1.374	1.243	0.131	3.1416
	1.379	1.253	0.126	3.1416
Valor Promedio	1.4834			

A continuación, se detalla resultado y análisis de programar, probar y ejecutar mismo cálculo aproximado del número Pi, pero esta vez utilizando framework de paralelización inspirado en esqueletos algorítmicos.

5.3.2. Programar y probar mismo algoritmo e-Science en framework y ejecutar en ciber infraestructura cluster MPI

La versión del algoritmo que calculó el valor aproximado del número Pi empleando el framework de paralelización, no utilizó explícitamente sentencias o instrucciones MPI. De acuerdo a decisiones de diseño del framework comentadas en resultados y análisis de actividades del segundo objetivo específico, el usuario que desarrolla sus algoritmos utilizando el framework no se ven en necesidad de emplear MPI directamente. Para desarrollar el algoritmo de cálculo del número Pi se utilizaron las mismas clases que se utilizaron en la programación del algoritmo en versión MPI pura, con la diferencia que estas operan dentro de la clase 'F' del framework.

En la clase F del framework, se le enmascara al usuario tareas de descomposición (en parte), distribución, comunicación y sincronización de procesos que generan ejecución de su algoritmo. En las diferentes funciones de la clase se ubica la lógica del algoritmo e-Science en cuestión de acuerdo con las interfaces que se le proporciona. En la operación 'prepareF' se inicializan algunos valores, entre estos un arreglo unidimensional que utiliza internamente la clase 'FDDC' para comunicar datos entre los nodos, y sobre los cuales estos pueden operar; queda a discreción del usuario que fin darle a este arreglo. La operación 'function' es aquella donde se especifica los cálculos individuales que hace cada nodo para obtener un valor, en este caso el valor aproximado de Pi. La operación 'functionPostScatter' se utiliza para aproximar aún más el cálculo de un valor calculado por los nodos de mayor nivel, en este caso Pi se sigue promediando al manipular el arreglo unidimensional que se menciona previamente; esta operación la llama cada nodo padre del cluster, excepto el nodo raíz. La operación 'functionMaster' la ejecutan sólo los nodos padres del cluster, y en este caso promedian aún más el valor de Pi. Finalmente, la operación 'functionFinal' la emplea el nodo raíz del cluster para realizar una última aproximación y exponer el resultado final del cálculo.

Se puede observar en las dos secciones de código fuente a continuación, la diferencia a nivel de código entre las dos versiones del algoritmo que calcula el valor aproximado del número Pi. Se puede apreciar que la versión MPI pura, que es la manera como tradicionalmente se escribe código distribuido, se escribe de manera explícita sentencias y variables MPI, mientras que en la versión del framework, estas no se utilizan.

Código 1: Archivo de código fuente ‘main.cpp’ (versión MPI Puro)

```
/*
 * main.cpp
 *
 *  Created on: Jan 10, 2014
 *      Author: mpiuser
 */
#include "mpi.h"
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <math.h>
#include "GeometricalManager.h"
using namespace std;

int main(int argc, char *argv[])
{
      int id;
      int numTasks;

      double finalPi=0;
      double averagePi=0;
      double localPi=0;
      double sumPi=0;
      double rounds=100;
      int numPoints=10000;
      int circleCount=0;
      double r;
      Circle circle;
      Rectangle rectangle;
      GeometricalManager gm;

      MPI_Init(&argc,&argv);
      MPI_Comm_size(MPI_COMM_WORLD, &numTasks);
      MPI_Comm_rank(MPI_COMM_WORLD, &id);

      r=1;
      circle.setRadius(r);
      rectangle.setLength(2*r);
```

```
	cout<<"MPI task "<<id<<" has started\n";


	for(int i=0;i<rounds;i++)
	{
		localPi=gm.computePi(r,numPoints,circleCount,rectangle);


	MPI_Reduce(&localPi,&sumPi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WO
RLD);

			if(id==0)
			{
				finalPi=sumPi/numTasks;
				averagePi=((averagePi * i) + finalPi)/(i + 1);
			}
	}

	if(id==0)
	{
			circle.setPi(averagePi);
			circle.setArea();
			cout<<"Number of points used is "<<numPoints<<"\n";
			cout<<"Circle's radius is "<<circle.getRadius()<<"\n";
			cout<<"Average value of Pi is "<<circle.getPi()<<"\n";
			cout<<"The average value of the circle's area is
"<<circle.getArea()<<"\n";
	}

	MPI_Finalize();
	return 0;

}
```

Código 2: Archivo de código fuente ‘F.h’ (versión framework)

```
/*
 * F.cpp
 *
 *  Created on: Jan 20, 2015
 *      Author: mpiuser
 */
#include <iostream>
#include <math.h>
#include "mpi.h"
#include "GeometricalManager.h"
using namespace std;


int numTasks=3;//;El arbol tiene 9 procesadores, de los cuales 6
ejecutan el algoritmo, 3 procesadores por rama del arbol
double averagePi=0.0;
double rounds=100;
int numPoints=10000;
int circleCount=0;
double r=1;
Circle circle;
Rectangle rectangle;
GeometricalManager gm;
//estos dos elementos son esenciales en el algoritmo
int numElementsInputArray=12;
double inputArray[12];

void prepareF()
{
      circle.setRadius(r);
      rectangle.setLength(2*r);
      inputArray[0]=10.0;inputArray[1]=10.0;inputArray[2]=10.0;inpu
tArray[3]=10.0;inputArray[4]=10.0;inputArray[5]=10.0;inputArray[6]
=10.0;inputArray[7]=10.0;inputArray[8]=10.0;inputArray[9]=10.0;inp
utArray[10]=10.0;inputArray[11]=10.0;
}
```

```
double function(int rounds)
{
      double localPi=0;
      for(int i=0;i<rounds;i++)
      {

      localPi+=gm.computePi(r,numPoints,circleCount,rectangle);
      }
      //cout<<"localPi: "<<localPi/rounds<<"\n";
      return localPi/rounds;

}

double functionMaster(int rank, int n, double gatherArray[])
{
      double sum=0;
      for(int i=0;i<n;i++)
      {
            sum+=gatherArray[i];
      }
      if(rank==0)
      {
            averagePi=sum/numTasks;
      }
      //cout<<"averagePi: "<<averagePi<<"\n";
      return averagePi;
}

void functionFinal(int rank,double val1, double val2)
{

      averagePi=(val1+val2)/2;
      if(rank==0)
      {
            circle.setPi(averagePi);
            circle.setArea();
            cout<<"Circle's radius is "<<circle.getRadius()<<"\n";
            cout<<"Average value of Pi is "<<circle.getPi()<<"\n";
            cout<<"The average value of the circle's area is
"<<circle.getArea()<<"\n";
      }
```

```
}

double functionPostScatter(int termCount, double localSum[])
{
      double sum=0;
      for(int i=0;i<termCount;i++)
      {
            sum+=function(localSum[i]);
      }
      sum=sum/termCount;
      return sum;
}
```

Se realizaron 10 ejecuciones del algoritmo en iguales condiciones de ejecución del algoritmo en versión framework, se midió el tiempo de ejecución en términos de tiempo de ‘usuario’ y de ‘sistema’ al igual que en la actividad anterior.

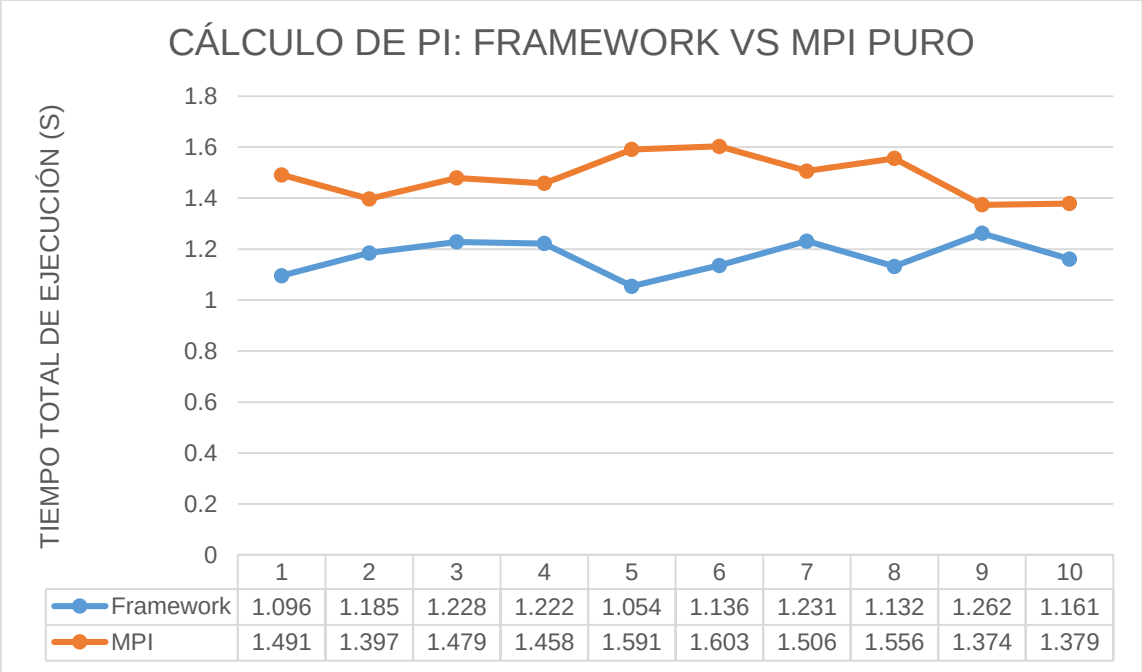
Para calcular el tiempo total de ejecución o procesamiento del algoritmo, se sumaron tanto el tiempo de ejecución de ‘usuario’, como el de ‘sistema’. En la Tabla 2 se aprecia resultados de las ejecuciones del algoritmo de cálculo aproximado del número Pi utilizando framework de paralelización.

Tabla 2: Tiempos de ejecución de cálculo aproximado del número Pi utilizando framework inspirado en esqueletos algorítmicos

	Tiempo Total de CPU	**Tiempo Usr (s)**	**Tiempo Sys (s)**	**Pi**
	1.096	0.858	0.238	3.1416
	1.185	0.901	0.284	3.1416
	1.228	0.982	0.246	3.1416
	1.222	0.915	0.307	3.1416
	1.054	0.822	0.232	3.1416
	1.136	0.853	0.283	3.1416
	1.231	0.87	0.361	3.1416
	1.132	0.808	0.324	3.1416
	1.262	0.992	0.27	3.1416
	1.161	0.921	0.24	3.1416
Valor Promedio	1.1707			

De acuerdo a Tablas 1 y 2, se realizó un comparativo entre los tiempos de procesamiento aproximado que tardan los algoritmos en calcular el valor aproximado de Pi. Como se puede apreciar en la Gráfica 1, la versión framework demora, en promedio, menos que su contraparte en MPI puro.

Gráfica 1: Ilustración comparativa de los tiempos de ejecución de ambos algoritmos



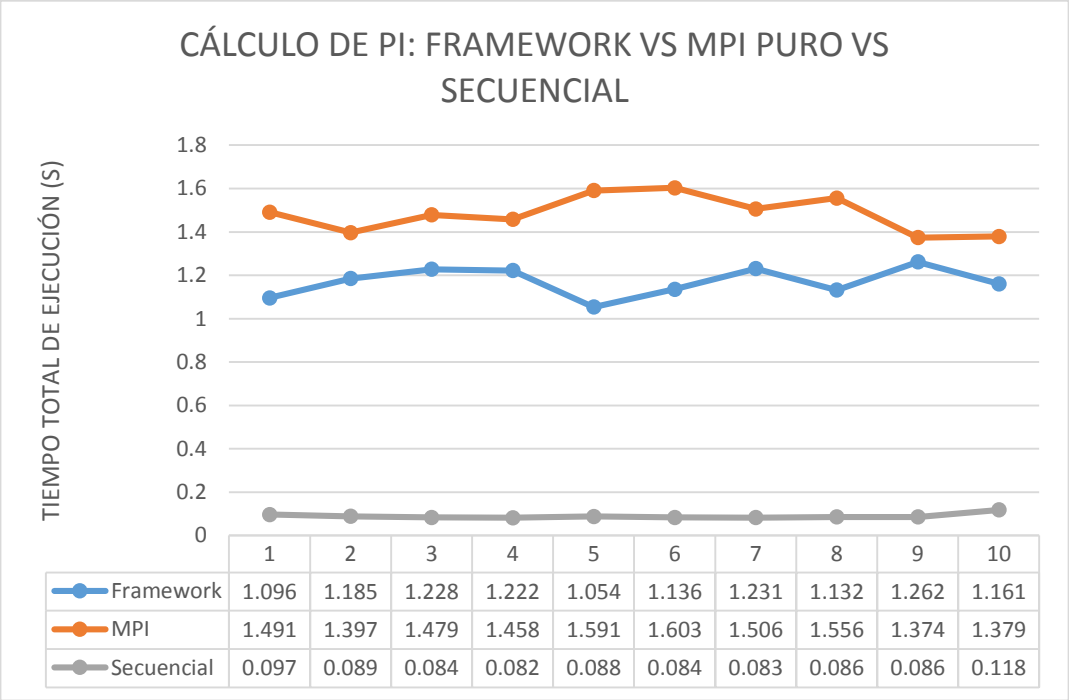
Adicionalmente, se realizó este mismo ejercicio de 10 ejecuciones del algoritmo en iguales condiciones que las ejecuciones del algoritmo en versión framework y versión MPI, pero en este caso se empleó una versión serial/secuencial del algoritmo del cálculo del número de Pi. La implementación hizo uso de un solo núcleo de procesamiento; esta puede ser revisada en el Anexo 07. La Tabla 3 muestra los resultados de las ejecuciones de esta versión del algoritmo.

Tabla 3: Tiempos de ejecución de cálculo aproximado del número Pi utilizando versión serial del algoritmo

	Tiempo Total de CPU	Tiempo Usr (s)	Tiempo Sys (s)	Pi
	0.097	0.093	0.004	3.1416
	0.089	0.089	0	3.1416
	0.084	0.084	0	3.1416
	0.082	0.082	0	3.1416
	0.088	0.088	0	3.1416
	0.084	0.084	0	3.1416
	0.083	0.079	0.004	3.1416
	0.086	0.086	0	3.1416
	0.086	0.086	0	3.1416
	0.118	0.114	0.004	3.1416
Valor Promedio	0.0897			

También, se realizó una ilustración comparativa con el promedio de tiempos de ejecuciones de las tres versiones del algoritmo del cálculo del número Pi, es decir MPI puro, framework y serial/secuencial. En Gráfico 2 se puede apreciar dicha gráfica, y en la cual se visualiza como la versión serial es mucho más rápida que las otras dos que utilizan ejecución distribuida.

Gráfica 2: Ilustración comparativa de los tiempos de ejecución de las tres versiones del algoritmo



	1	2	3	4	5	6	7	8	9	10
Framework	1.096	1.185	1.228	1.222	1.054	1.136	1.231	1.132	1.262	1.161
MPI	1.491	1.397	1.479	1.458	1.591	1.603	1.506	1.556	1.374	1.379
Secuencial	0.097	0.089	0.084	0.082	0.088	0.084	0.083	0.086	0.086	0.118

A continuación, se detalla resultado y análisis de calcular múltiples métricas de complejidad respecto a las ejecuciones de ambos algoritmos ya desarrollados y probados. Se busca con dichas métricas establecer cuál de las dos versiones del algoritmo proporciona mayores facilidades, o brinda mayor provecho de desarrollo para el usuario final.

5.3.3. Calcular y analizar valores de métricas de complejidad para ambos algoritmos

Con el objetivo de establecer referencias para determinar si framework de paralelización proporcionó mayor provecho de ciber infraestructura para desarrollo de algoritmos, se registraron métricas que permitiesen valorar si esta afirmación es en efecto verdadera, o no, para el caso concreto del cálculo aproximado del número

Pi. En Tablas 3 y 4 a continuación, para ambos algoritmos desarrollados en actividades previas, se especifica métricas de Líneas Lógicas de Código Fuente (LLOC), Longitud del Programa (N), Vocabulario del Programa (n), Volumen del Programa (V), Nivel de Dificultad del programa (D), Nivel del Programa (L), Esfuerzo de Implementación (E), Tiempo de Implementación (T), Suma de Complejidad Ciclomática (Sum(CC)), Conteo de Complejidad Ciclomática (Count(CC)), Total de Complejidad Ciclomática (Total(CC)), e Índice de Mantenibilidad (MI). Para el cálculo de las métricas se hizo uso de archivo de contiene código de implementación de algoritmo e-Science en versión MPI puro y se denomina 'main', mientras que el archivo que contiene código fuente de la implementación del algoritmo e-Science por medio de framework se denomina 'F'.

Tabla 4: Métricas de complejidad para algoritmo de cálculo aproximado de valor Pi utilizando MPI puro.

LLOC	**n1**	**n2**	**N1**	**N2**	**Compl. Ciclomática (CC)**	**Índice de Mantenibilidad (MI)**
50	27	60	168	135	4	80.79285
Longitud del Programa (N)	**Vocabulario del Programa (n)**	**Volumen del Programa (V) en bits y bytes**	**Nivel de Dificultad (D)**	**Nivel del Programa (L)**	**Esfuerzo de Implementar (E)**	**Tiempo en Implementar (T) en seg, min y h**
303	87	1952.212	30.375	0.032922	59298.44	3294.358
		244.0265				54.90596
						0.915099

Tabla 5: Métricas de complejidad para algoritmo de cálculo aproximado de valor Pi utilizando framework de paralelización inspirado en esqueletos algorítmicos.

LLOC	n1	n2	N1	N2	Compl. Ciclomática (CC)	Índice de Mantenibilidad (MI)
66	25	62	215	181	6	74.92839345
Longitud del Programa (N)	**Vocabulario del Programa (n)**	**Volumen del Programa (V) en bits y bytes**	**Nivel de Dificultad (D)**	**Nivel del Programa (L)**	**Esfuerzo de Implementar (E)**	**Tiempo en Implementar (T) en seg, min y h**
395	87	2544.963	36.49194	0.027403	92870.61	5159.478553
		318.1203				85.99130922
						1.433188487

Las LLOC son el número total de líneas de código fuente, exceptuando líneas en blanco y líneas comentadas. Se emplea comúnmente para medir el tamaño de un programa, y dependiendo de qué tan alto o bajo sea el valor, ello indica que tan complejo y cuanto trabajo conlleva desarrollar una determinada pieza software. Se sugiere como máximo número aceptable de LLOC en un archivo de programa software el valor de 2000 [53].

Las métricas Halstead N, n, V, D, L, E y T son valores obtenidos del código fuente de una pieza software, con los que se determina que tan complejo es dicha pieza. Dichos valores se obtienen de la cantidad de operandos y operadores del código fuente. Los operadores son generalmente símbolos, estructuras de control y

palabras reservadas, aunque esto varía con el lenguaje de programación y también de acuerdo a las apreciaciones particulares de la academia y la industria, ya que no existe estándar definido de que exactamente cuento o no como un operador. Los operandos son constantes, identificadores y especificadores de tipos, y así como para caso de los operadores, no existe estándar que estipule exactamente que puede ser o no ser un operando. Halstead determina a partir de esto, Número Único de Operadores (n1), Número Único de Operandos (n2), Número Total de Operadores (N1) y Número Total de Operandos (N2) [22] [49]. Los operadores y operandos contabilizados en los algoritmos desarrollados en las actividades previas a esta pueden apreciarse en detalle en el anexo 06, dicha cuenta fue realizada de forma manual en ambos algoritmos y utilizando mismo criterios de decisión.

Con base en n1, n2, N1 y N2, se obtiene varias métricas Halstead: La Longitud del Programa (N), donde N = N1 + N2. El Tamaño del Vocabulario (n), donde n = n1 + n2. El Volumen de Programa (V), donde V = N * Log2(n), y mide en bits el tamaño de implementación del programa. El volumen se mide con base en el número de operaciones realizadas y número de operandos que este maneja, por lo que de acuerdo a Halstead esta métrica es menos sensible a la estructura del código que la métrica LLOC. También, el volumen de un archivo debe ser mínimo 100 y máximo 8000 [22] [49].

El Nivel de Dificultad del programa (D), donde D = (n1/2) * (N2/n2), establece grado de dificultad, o que tan propenso es el código a generar/introducir errores, y esto es directamente proporcional al número de operadores únicos del programa y a la razón del número total de operandos y el número único de operandos, es decir, si se reutiliza los mismos operandos en el código, más este se encuentra expuesto a introducir errores. El Nivel del Programa (L), L = 1/D, es la inversa de la dificultad o exposición a errores, es decir, un algoritmo de bajo nivel es más propenso a errores que uno de alto nivel. El Esfuerzo de Implementación (E), E = V * D, permite determinar qué tan fácil de implementar o comprender una pieza software. Finalmente, Halstead propone que el Tiempo aproximado de Implementar un programa (T), se determina por T = E/18, en donde la constante 18 expone el valor en segundos [22] [49].

Las métricas de McCabe, agrupadas bajo la Complejidad Ciclomática (CC), matemáticamente establecida como V(g), son otro conjunto de medidas que determinan el grado de complejidad de piezas software con base en decisiones explícitas en el código fuente del software. La contabilización de las decisiones se basa en sentencias condicionales del código fuente [42]. La Complejidad Ciclomática se calcula como CC = número de decisiones +1, donde la constante 1 se contabiliza como inicio de la sección que contiene el código fuente. La CC se contabiliza por operaciones individuales pero se puede hallar para un programa entero. Un alto grado de CC indica alta dificultad para comprender una pieza software, alta exposición a errores, gran cantidad de tiempo de implementación y pruebas. Similar al caso de las métricas Halstead con los operadores y operandos, dependiendo del lenguaje de programación, y criterios académicos e industriales, el criterio para determinar que estructura es una decisión o no, puede variar ligeramente [42].

Una pieza software con valor CC de, 1 < CC < 4, indica bajo riesgo para mantener, comprender y probar, mientras que una pieza software de CC > 50 indica muy alto riesgo, y si CC > ya se considera como una situación alarmante. Para obtener el valor CC Total de un programa se debe considerar la ecuación: Total(CC) = Sum(CC) - Count(CC) + 1, donde Count(CC) es la cuenta del número total de procedimiento u operaciones del algoritmo/programa y Sum(CC) es la suma total de todos los CC de cada operación del algoritmo [42]. En figuras 34 y 35 se puede apreciar la contabilización de las decisiones que conforman el cálculo de la Complejidad Ciclomática para algoritmos que calculan valor aproximado del número Pi empleando MPI puro y framework de paralelización inspirado en esqueletos algorítmicos; en estos algoritmos se utilizan estructuras lógicas del control muy básicas como lo son la ‘if’ y ‘for’.

Figura 34: Contabilización de decisiones que conforman cálculo de la Complejidad Ciclomática para algoritmos que calculan valor aproximado del número Pi empleando MPI puro

```
/*
 * main.cpp
 *
 *  Created on: Jan 10, 2014
 *      Author: mpiuser
 */
#include "mpi.h"
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <math.h>
#include "GeometricalManager.h"
using namespace std;

int main(int argc, char *argv[])    1
{
        int id;
        int numTasks;

        double finalPi=0;
        double averagePi=0;
        double localPi=0;
        double sumPi=0;
        double rounds=100;
        int numPoints=10000;
        int circleCount=0;
        double r;
        Circle circle;
        Rectangle rectangle;
        GeometricalManager gm;

        MPI_Init(&argc,&argv);
        MPI_Comm_size(MPI_COMM_WORLD, &numTasks);
        MPI_Comm_rank(MPI_COMM_WORLD, &id);

        r=1;
        circle.setRadius(r);
        rectangle.setLength(2*r);

        cout<<"MPI task "<<id<<" has started\n";

                                        1
        for(int i=0;i<rounds;i++)
        {
                localPi=gm.computePi(r,numPoints,circleCount,rectangle);

                MPI_Reduce(&localPi,&sumPi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);

                                        1
                if(id==0)
                {
                        finalPi=sumPi/numTasks;
                        averagePi=((averagePi * i) + finalPi)/(i + 1);
                }
        }
                                        1
        if(id==0)
        {
                circle.setPi(averagePi);
                circle.setArea();
                cout<<"Number of points used is "<<numPoints<<"\n";
                cout<<"Circle's radius is "<<circle.getRadius()<<"\n";
                cout<<"Average value of Pi is "<<circle.getPi()<<"\n";
                cout<<"The average value of the circle's area is "<<circle.getArea()<<"\n";
        }

        MPI_Finalize();
        return 0;

}
```

Figura 35: Contabilización de decisiones que conforman cálculo de la Complejidad Ciclomática para algoritmos que calculan valor aproximado del número Pi empleando framework de paralelización inspirado en esqueletos algorítmicos

```
/*
 * F.cpp
 *
 *  Created on: Jan 20, 2015
 *      Author: mpiuser
 */
#include <iostream>
#include <math.h>
#include "mpi.h"
#include "GeometricalManager.h"
using namespace std;


int numTasks=3;//;El arbol tiene 9 procesadores, de los cuales 6 ejecutan el algoritmo, 3 procesadores por rama del arbol
double averagePi=0.0;
double rounds=100;
int numPoints=10000;
int circleCount=0;
double r=1;
Circle circle;
Rectangle rectangle;
GeometricalManager gm;
//estos dos elementos son esenciales en el algoritmo
int numElementsInputArray=12;
double inputArray[12];

void prepareF()    1
{
        circle.setRadius(r);
        rectangle.setLength(2*r);
        inputArray[0]=10.0;inputArray[1]=10.0;inputArray[2]=10.0;inputArray[3]=10.0;inputArray[4]=10.0;inputArray[5]=10.0;inputArray[6]=10.0;inputArray[7]=10.0;inputArray[8]=10.0;inputArray[9]=10.0;inputArray[10]=10.0;inputArray[11]=10.0;
}


double function(int rounds)      1
{
        double localPi=0;
        for(int i=0;i<rounds;i++)      1
        {
                localPi+=gm.computePi(r,numPoints,circleCount,rectangle);
        }
        //cout<<"localPi: "<<localPi/rounds<<"\n";
        return localPi/rounds;

}

double functionMaster(int rank, int n, double gatherArray[])        1
{
        double sum=0;
        for(int i=0;i<n;i++)          1
        {
                sum+=gatherArray[i];
        }
        if(rank==0)                   1
        {
                averagePi=sum/numTasks;
        }
        //cout<<"averagePi: "<<averagePi<<"\n";
        return averagePi;
}

void functionFinal(int rank,double val1, double val2)        1
{

        averagePi=(val1+val2)/2;
        if(rank==0)                   1
        {
                circle.setPi(averagePi);
                circle.setArea();
                cout<<"Circle's radius is "<<circle.getRadius()<<"\n";
                cout<<"Average value of Pi is "<<circle.getPi()<<"\n";
                cout<<"The average value of the circle's area is "<<circle.getArea()<<"\n";
        }
}

double functionPostScatter(int termCount, double localSum[])      1
{
        double sum=0;
        for(int i=0;i<termCount;i++)                  1
        {
                sum+=function(localSum[i]);
        }
        sum=sum/termCount;
        return sum;
}
```

Finalmente, Oman y Hagemeister proponen la métrica de complejidad conocida como el Índice de Mantenibilidad (MI), el cual mide que tan facil se presupone es mantener una pieza software. Esta métrica es avalada por sectores académicos y de la industria de tecnologías de la información [43] [50]. La métrica MI se obtiene

de, MI = 171 - 3.42*Ln(aveE) - 0.23*aveCC - 16.2*Ln(aveLOC), donde para una pieza o módulo software, aveE corresponde el valor aproximado del Esfuerzo de Implementación de Halstead, aveV(g) es el valor aproximado de la Complejidad Ciclomática y aveLOC es el número aproximado de Líneas de Código, pero que en este trabajo se precisa con uso de las Líneas Lógicas de Código LLOC [50]. Se sugiere como valores óptimos de mantenibilidad para el Índice de Mantenibilidad: 20 < MI < 100 [53]. En las figuras 36, 37, 38 y 39 a continuación, se puede apreciar las fórmulas de las que se obtuvieron las métricas Halstead, de McCabe y el Indice de Mantenibilidad.

Figura 36: Métricas de Complejdad de Halstead

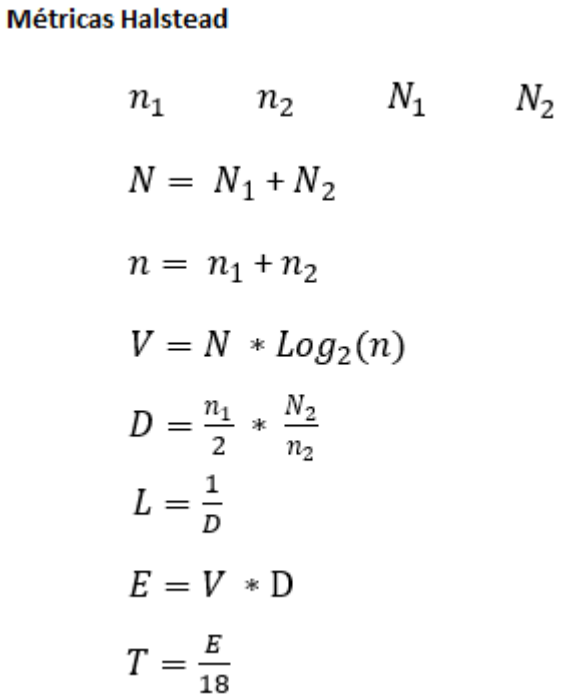


Figura 37: Métricas de McCabe

Complejidad Ciclomática

$$CC = No.de\ decisiones + 1$$

$$Total(CC) = Sum(CC) - Count(CC) + 1$$

Figura 38: Métrica de Líneas Lógicas de Código

Otras Métricas

$$LLOC$$

Figura 39: Métrica de Oman y Hagemeister

Indice de Mantenibilidad

$$MI = 171 - 3.42 * \mathrm{LN}(\mathrm{aveE}) - 0.23 * \mathrm{aveCC} - 16.2 * \mathrm{LN}(\mathrm{aveLOC})$$

De acuerdo a Tablas 3 y 4, se puede apreciar que el valor LLOC fue mayor en la versión del algoritmo desarrollado con MPI puro, respecto a la versión que empleó el framework, esto sucedió porque la versión framework está condicionada ya que es una plantilla con código fuente predefinido, y en condiciones iniciales parte con líneas lógicas de código fuente de más. A nivel de cantidad de operadores y operandos únicos, ambas versiones del algoritmo manejaron valores casi que iguales, sin embargo esa paridad se inclinó a favor de la versión MPI que no utilizó

tantas repeticiones de estos, y fué en detrimento de la versión framework, ya que al forzar utilización e inicialización del arreglo unidimensional que no utilizó la versión MPI, generó desbalance en sus métricas respecto a su algoritmo rival.

A raíz de la utilización predeterminada de este arreglo en el framework, se pudo apreciar como los valores de las métricas Halstead tiendieron a favorecer al algoritmo en versión MPI puro. Valores como Longitud, Volumen, Nivel de Programa, Nivel de Dificultad, Esfuerzo y Tiempo de implementación, favorecieron la implementación MPI pura por muy poco, a excepción de la métrica de Esfuerzo de Implementación. La naturaleza propia del framework también condicionó los valores de Complejidad Ciclomática, donde se evidenció menor complejidad en el algoritmo MPI puro. Finalmente, el Índice de Mantenibilidad también favorecío la implementación MPI pura, sin embargo los valores MI de ambos algoritmos son adecuados.

Antes de finalizar este capítulo, exponemos las métricas de complejidad obtenidas para el algoritmo que calcula el valor aproximado del número Pi de forma serial/secuencial empleando un solo procesador/núcleo. Todos los valores de complejidad son mejores que sus contrapartes, es decir, el algoritmo en versiones MPI puro y Framework.

Tabla 6: Métricas de complejidad para algoritmo de cálculo aproximado de valor Pi utilizando programación secuencial/serial.

LLOC	n1	n2	N1	N2	Compl. Ciclomática (CC)	Índice de Mantenibilidad (MI)
34	23	43	92	89	2	89.48107078
Longitud del Programa (N)	**Vocabulario del Programa (n)**	**Volumen del Programa (V) en bits y bytes**	**Nivel de Dificultad (D)**	**Nivel del Programa (L)**	**Esfuerzo de Implementar (E)**	**Tiempo en Implementar (T) en seg, min y h**
181	66	1094.035336	23.80232558	0.042012702	26040.58526	1446.699181
		136.754417				24.11165301
						0.401860884

Con base en el desarrollo, exposición y análisis de las actividades de este trabajo, se procede a establecer algunas conclusiones en referencia a premisas planteadas al inicio del mismo.

CAPÍTULO VI: CONCLUSIONES

En esta sección final del escrito, se presentan múltiples conclusiones que se derivan de los resultados de la ejecución de las actividades de este trabajo. Las conclusiones se presentan por orden de los objetivos específicos.

Se especificó la arquitectura software del ecosistema de aprovechamiento, la cual recolectó requerimientos funcionales, atributos de calidad, modelos estáticos y de comportamiento del mismo, y determinó directrices de implementación del ecosistema de aprovechamiento. La arquitectura se implementó en gran proporción y de forma parcial por medio de la ejecución de todas las actividades descritas en este trabajo. Se habla de implementación parcial de la arquitectura ya que en el modelo despliegue, algunos requerimientos, en cuanto a atributos de calidad y algunas características iniciales del diseño del framework de paralelización, no lograron ser completamente aplicadas al término de este trabajo, por el tiempo que requiere su total implementación. Sin embargo, se especificó la arquitectura concreta del ecosistema HPC, que puede ser materializada a plenitud, revisada y modificada en trabajos posteriores, debido a que todas las tecnologías y conceptos empleadas son de libre acceso y ampliamente documentados.

Por otro lado, las actividades del primer objetivo específico, resultaron en que un usuario final cualquiera del ecosistema pudiera emplear tecnologías como lenguajes de programación y compiladores/interpretadores para C, C++, Java, Python y FORTRAN, otras herramientas y extensiones de desarrollo como IDEs, hilos, paso de mensajes, HTCondor, herramientas de monitoreo como Munin y otras implícitas en el sistema operativo basado en Linux Red Hat; también otras como plataformas de despliegue virtual de nodos y variadas tecnologías basadas en redes, que facilitan la reproducción, mantenimiento, escalación y mejora del ecosistema. Todo este conjunto de diversas y potentes características posibilitó mejor aprovechamiento de las ciber infraestructuras hardware y software existentes en la UTB, lo cual responde en gran medida la pregunta de investigación planteada al inicio del escrito.

En el marco de la hipótesis planteada al inicio de este escrito, se afirma que el aprovechamiento de estas ciber infraestructuras HPC si proporciona mayores alternativas y facilidades para el desarrollo y ejecución de tareas e-Science, esto se afirma con base en las experiencias similares expuestas por instituciones como UniAndes, Polito, CERN, Universidad de huddersfield, y actores gestores del ecosistema en la UTB, con lo cual parte de esta hipótesis es valedera.

Se logró desarrollar bases del framework de programación paralela inspirado en concepto de esqueletos algorítmicos para resolver problemas e-Science por reducción de datos. Se describieron aspectos de diseño a tener en cuenta para implementar una solución de esta índole; se tomaron decisiones con base en estas mismas directrices expresadas en la literatura, y comunidades dedicadas a la explotación de la programación paralela distribuida, y se ajustaron a necesidades propias de proporcionar herramienta flexible, que sirva de alternativa y facilite a usuarios el desarrollo de sus algoritmos e-Science por reducción de datos que requieren de HPC.

El diseño del framework de paralelización inspirado en esqueletos algorítmicos condicionó los niveles de esfuerzo y complejidad para el caso de un algoritmo e-Science que aproxima el valor del número Pi. A nivel de desempeño, este algoritmo en su versión framework se procesó en mejores tiempos, es decir menores, que su contraparte en MPI puro, mientras que los valores en métricas de esfuerzo de implementación y complejidad en general se ven algo desfavorecidos. Estas cuantificaciones en parte socavan la veracidad de la hipótesis de este trabajo, ya que, para esta implementación de aproximación del número Pi que se desarrolló, la versión framework no facilita el desarrollo de este algoritmo, debido a que su código y estructura de datos predefinida no influyeron de forma positiva al momento de calcular las métricas pertinentes. Aun así, es importante señalar que dicho código y estructura de datos predefinida, se planteó de esa manera para ofrecer al usuario final mayor flexibilidad al momento de escribir su algoritmo sobre el framework, a diferencia de la utilización fiel del concepto esqueletos algorítmicos, ya que estos limitan control al usuario sobre su algoritmo si no se cuenta con una alta gama y variedad de estos. Por otra parte, también es válido afirmar, en referencia a la hipótesis de este trabajo, que el framework que se desarrolló para esta tesis es una

herramienta alternativa que potencia el aprovechamiento de las demás ciber infraestructuras desplegadas en este trabajo, respaldado esto por el hecho que se logró mejorar los tiempos de ejecución de un algoritmo e-Science empleando el framework de paralelización.

Para finalizar, la pregunta de investigación planteada en este trabajo: ¿Qué características de un ecosistema HPC permiten mejor aprovechamiento de ciber infraestructura hardware y software existente?', se responde: con pertinente arquitectura de sistema, (materializada a través de), tecnologías de virtualización, comunicación remota, sistemas de compilación, desarrollo y ejecución de algoritmos, sistemas de navegación y almacenamiento de archivos, todas estas de libre de distribución. Y Adicionalmente, con framework software inspirado en esqueletos algorítmicos para el desarrollo de algoritmos e-Science por reducción de datos, que aunque en este trabajo no se logra confirmar cuantitativamente y a través de una aplicación en particular, las facilidades que estos brindan a sus usuarios en sus implementaciones, otros autores exponen sobre el valor agregado que estos generan a sus usuarios al aprovechar ciber infraestructuras hardware y software existentes.

6.1. TRABAJO FUTURO

En un futuro se plantea extender este trabajo. Se plantea realizar una medida del aprovechamiento de ciber infraestructuras HPC usando framework, ya no con métricas cuantitativas, sino con medidas cualitativas, es decir, se busca que mismos usuarios finales, estos son, estudiantes, profesores, investigadores y otros, sean quienes determinen si dicha framework proporciona facilidades al momento de desarrollar sus algoritmos e-Science. Así mismo, establecer por esa misma vía, si estos usuarios consideran que los despliegues de ciber infraestructura desarrollados en este trabajo hacen un eficaz aprovechamiento de la ciber infraestructura UTB donde se desplegaron. Adicionalmente, proponer implementar framework de paralelización usando todas las decisiones de diseño planteadas por Cole, es decir uso de funciones de orden superior, incorporar todos sus esqueletos, e inclusive plantear nuevos esqueletos, esto para abarcar mayor número de problemas e-Science, y seguir disminuyendo los niveles de complejidad de programación; también, realizar comparativos respecto a la versión anterior del

framework. También se plantea portar el framework a otros lenguajes de programación, que el ecosistema proporcione procesamiento HPC por medio de GPUs, extender la capacidad de la ciber infraestructura empleando más tecnologías en nube, realizar pruebas de uso de memoria del ecosistema, evaluar como el ecosistema de aprovechamiento se comporta en el marco de políticas de 'Green IT', entre otros.

REFERENCIAS

[1] W. W. Smari, S. Fiore, and D. Hill, “High performance computing and simulation: architectures, systems, algorithms, technologies, services, and applications,” Concurrency and Computation: Practice and Experience, 2013.

[2] M. Riedel, F. Wolf, D. Kranzlmüller, A. Streit, and T. Lippert, “Research advances by using interoperable e-science infrastructures,” Cluster Computing, vol. 12, no. 4, pp. 357–372, 2009.

[3] H. Castro, E. Rosales, M. Villamizar, and A. Jiménez, “UnaGrid: On Demand Opportunistic Desktop Grid,” in Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010, pp. 661–666.

[4] N. Nepote, E. Piccolo, C. G. Demartini, and P. Montuschi, “Why and How Using HPC in University Teaching? A Case Study at PoliTo.”

[5] C. L. Jiménez, “Datagrid: Infraestructura de computación en malla para integración de aplicaciones y de información,” Revista Sistemas ACIS, edición, vol. 98, 2007.

[6] E. C. Dragut, P. Baker, J. Xu, M. I. Sarfraz, E. Bertino, A. Madhkour, R. Agarwal, A. Mahmood, and S. Han, “CRIS—Computational research infrastructure for science,” in Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on, 2013, pp. 301–308.

[7] V. V. Kindratenko, J. J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E. Stone, J. C. Phillips, and W. Hwu, “GPU clusters for high-performance computing,” in Cluster Computing and Workshops, 2009. CLUSTER’09. IEEE International Conference on, 2009, pp. 1–8.

[8] Y. Zhai, M. Liu, J. Zhai, X. Ma, and W. Chen, “Cloud versus in-house cluster: evaluating Amazon cluster compute instances for running MPI applications,” in State of the Practice Reports, 2011, p. 11.

[9] R. Rabenseifner, G. Hager, and G. Jost, “Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes,” in Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on, 2009, pp. 427–436.

[10] A. Gorobets, F. X. Trias, R. Borrell, O. Lehmkuhl, and A. Oliva, “Hybrid MPI+ OpenMP parallelization of an FFT-based 3D Poisson solver with one periodic direction,” Computers & Fluids, vol. 49, no. 1, pp. 101–109, 2011.

[11] Bp. Gonzalez, J. P. Donate, P. Cortez, Gg. Sánchez, and As. De Miguel, “Parallelization of an evolving Artificial Neural Networks system to Forecast Time Series using OPENMP and MPI,” in Evolving and Adaptive Intelligent Systems (EAIS), 2012 IEEE Conference on, 2012, pp. 186–191.

[12] B. J. Pope, B. G. Fitch, M. C. Pitman, J. J. Rice, and M. Reumann, “Performance of hybrid programming models for multiscale cardiac simulations: Preparing for petascale computation,” Biomedical Engineering, IEEE Transactions on, vol. 58, no. 10, pp. 2965–2969, 2011.

[13] J. Murong, C. Lin, Z. Jun, W. Dan, and T. Fang, “Parallel Image Registration with OpenMP and MPI,” in Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on, 2009, pp. 1–4.

[14] M. D. Jones and R. Yao, “Parallel programming for OSEM reconstruction with MPI, OpenMP, and hybrid MPI-OpenMP,” in Nuclear Science Symposium Conference Record, 2004 IEEE, 2004, vol. 5, pp. 3036–3042.

[15] S. McConnell, R. Sturgeon, G. Henry, A. Mayne, and R. Hurley, “Scalability of Self-organizing Maps on a GPU cluster using OpenCL and CUDA,” in Journal of Physics: Conference Series, 2012, vol. 341, p. 012018.

[16] R. Xu, S. Chandrasekaran, and B. Chapman, “Exploring Programming Multi-GPUs Using OpenMP and OpenACC-Based Hybrid Model,” in Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International, 2013, pp. 1169–1176.

[17] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, and J. Wawrzynek, “A view of the parallel computing landscape,” Communications of the ACM, vol. 52, no. 10, pp. 56–67, 2009.

[18] W. Xu, Y. Wu, W. Xue, W. Zhang, Y. Yuan, and K. Zhang, “Horde: A parallel programming framework for clusters,” in Web Society, 2009. SWS’09. 1st IEEE Symposium on, 2009, pp. 96–101.

[19] J. Falcou, “Parallel Programming with Skeletons,” Computing in Science & Engineering, vol. 11, no. 3, pp. 58–63, 2009.

[20] R. Arora and P. Bangalore, “A framework for raising the level of abstraction of explicit parallelization,” in Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on, 2009, pp. 339–342.

[21] Legaux, J.; Loulergue, F.; Jubertie, S., "Development effort and performance trade-off in high-level parallel programming," High Performance Computing & Simulation (HPCS), 2014 International Conference on , vol., no., pp.162,169, 21-25 July 2014

[22] Shen, V.Y.; Conte, S.D.; Dunsmore, H.E., "Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support," Software Engineering, IEEE Transactions on , vol.SE-9, no.2, pp.155,165, March 1983

[23] De Silva, D.I.; Kodagoda, N.; Perera, H., "Applicability of three complexity metrics," Advances in ICT for Emerging Regions (ICTer), 2012 International Conference on , vol., no., pp.82,88, 12-15 Dec. 2012

[24] Nick Rozanski and Eóin Woods. 2005. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives. Addison-Wesley Professional

[25] Sei.cmu.edu, 'Software Architecture | Tools & Methods | Documenting the Architecture | Views and Beyond: The SEI Approach to Architecture Documentation', 2015. [Online]. Available: http://www.sei.cmu.edu/architecture/tools/document/viewsandbeyond.cfm. [Accessed: 28- Feb- 2015].

[26] Mitpress.mit.edu, 'Beowulf Cluster Computing with Linux | The MIT Press', 2015. [Online]. Available: http://mitpress.mit.edu/books/beowulf-cluster-computing-linux. [Accessed: 28- Feb- 2015].

[27] D. Gubb, 'Implementation of A Condor Pool At The University Of Huddersfield That Conforms To A Green IT Policy', University of Huddersfield, 2013.

[28] Ursuleanu, M.-F.; Graur, A.; Dimian, M., "Developing cyberinfrastructure for advanced research with high performance computing," Roedunet International Conference (RoEduNet), 2010 9th , vol., no., pp.364,367, 24-26 June 2010

[29] Research.cs.wisc.edu, 'HTCondorTM Version 8.3.3 Manual', 2015. [Online]. Available: http://research.cs.wisc.edu/htcondor/manual/v8.3/. [Accessed: 28- Feb- 2015].

[30] Guide.munin-monitoring.org, 'Welcome to the Munin Guide — Munin 2.1.10-detached-2015-02-28-c92-g8820c16 documentation', 2015. [Online]. Available: http://guide.munin-monitoring.org/en/latest/. [Accessed: 01- Mar- 2015].

[31] Helene, C.; Sebastien, L., "Algorithmic skeleton library for scientific simulations: SkelGIS," High Performance Computing and Simulation (HPCS), 2013 International Conference on , vol., no., pp.429,436, 1-5 July 2013

[32] C. Murray, Algorithmic Skeletons: Structured Management of Parallel Computation. Cambridge, MA, USA: MIT Press, 1991.

[33] Nereida.deioc.ull.es (Universidad de la Laguna), 'Últimos Avances en Informática. UAI'2003', 2015. [Online]. Available: http://nereida.deioc.ull.es/~pcgull/lacs-uai03/slides/MurrayCole.pdf. [Accessed: 01- Mar- 2015].

[34] P. Ciechanowicz, M. Poldner and H. Kuchen, The Münster Skeleton Library Muesli: A comprehensive overview, 1st ed. Westfälsche Wilhelms-Universität Münster (WWU) - European Research Center for Information Systems (ERCIS), 2009.

[35] W. Gropp, MPI - the complete reference. Cambridge, Mass. [u.a.]: MIT Press, 1998.

[36] Real Academia Española, 'DIccionario de la Lengua Española', 2015. [Online]. Available: http://lema.rae.es/drae/?val=ecosistema. [Accessed: 28- Feb- 2015].

[37] Tiobe.com, 'TIOBE Software: The Coding Standards Company', 2015. [Online]. Available: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. [Accessed: 28- Feb- 2015].

[38] Oracle.com, 'Mixing C and C++ Code in the Same Program', 2015. [Online]. Available: http://www.oracle.com/technetwork/articles/servers-storage-dev/mixingcandcpluspluscode-305840.html. [Accessed: 28- Feb- 2015].

[39] Techopedia.com, 'What is Software Framework? - Definition from Techopedia', 2015. [Online]. Available: http://www.techopedia.com/definition/14384/software-framework. [Accessed: 28- Feb- 2015].

[40] H. Erdogmus and O. Tanir, ‘Advances in Software Engineering: Comprehension, Evaluation, and Evolution’, Springer, Jan. 2002.

[41] T. Ladan and A. Singh, ‘Impact of Using Pattern-Based Systems on the Qualities of Parallel Applications’, University of Waterloo, 2000.

[42] Aivosto.com, 'Project Metrics Help - Complexity metrics', 2015. [Online]. Available: http://www.aivosto.com/project/help/pm-complexity.html. [Accessed: 01- Mar- 2015].

[43] Coleman, D.; Ash, D.; Lowther, B.; Oman, P., "Using metrics to evaluate software system maintainability," Computer , vol.27, no.8, pp.44,49, Aug. 1994

[44] D. Sjøberg, B. Anda and A. Mockus, 'Questioning software maintenance metrics', Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '12, 2012.

[45] McCabe, T.J., "A Complexity Measure," Software Engineering, IEEE Transactions on , vol.SE-2, no.4, pp.308,320, Dec. 1976

[46] Aivosto.com, 'Project Metrics Help - Lines of code metrics (LOC)', 2015. [Online]. Available: http://www.aivosto.com/project/help/pm-loc.html. [Accessed: 01- Mar- 2015].

[47] M. Cole, Parallel Programming with Skeletons: Past, Present and Future, 1st ed. Edinburgo: Universidad de Edinburgo, 2003.

[48] Computing.llnl.gov, 'Message Passing Interface (MPI)', 2015. [Online]. Available: https://computing.llnl.gov/tutorials/mpi/. [Accessed: 03- Apr- 2015].

[49] Verifysoft.com, 'Verifysoft → Halstead Metrics', 2015. [Online]. Available: http://www.verifysoft.com/en_halstead_metrics.html. [Accessed: 05- Apr- 2015].

[50] Virtualmachinery.com, 'Virtual Machinery - Sidebar 4 - MI and MINC - Maintainability Index', 2015. [Online]. Available: http://www.virtualmachinery.com/sidebar4.htm. [Accessed: 05- Apr- 2015].

[51] A. Deursen, 'Think Twice Before Using the "Maintainability Index"', Arie van Deursen, 2014. [Online]. Available: http://avandeursen.com/2014/08/29/think-twice-before-using-the-maintainability-index/. [Accessed: 05- Apr- 2015].

[52] Aivosto.com, 'Project Metrics Help - Lines of code metrics (LOC)', 2015. [Online]. Available: http://www.aivosto.com/project/help/pm-loc.html. [Accessed: 05- Apr- 2015].

[53] Blogs.msdn.com, 'Code Metrics – Maintainability Index - The Ultimate Visual Studio Tips and Tricks Blog - Site Home - MSDN Blogs', 2015. [Online]. Available: http://blogs.msdn.com/b/zainnab/archive/2011/05/26/code-metrics-maintainability-index.aspx. [Accessed: 05- Apr- 2015].

[54] Kb.iu.edu, 'What is cyberinfrastructure?', 2015. [Online]. Available: https://kb.iu.edu/d/auhf. [Accessed: 06- Apr- 2015].

[55]J. Syed, M. Ghanem and Y. Guo, 'Supporting scientific discovery processes in Discovery Net', Concurrency Computat.: Pract. Exper., vol. 19, no. 2, pp. 167-179, 2006.

[56] Escience.washington.edu, 'eScience Team | eScience Institute', 2015. [Online]. Available: http://escience.washington.edu/who-we-are/escience-team. [Accessed: 08- Apr- 2015].

[57] insideHPC, 'What is high performance computing? - insideHPC', 2009. [Online]. Available: http://insidehpc.com/hpc-basic-training/what-is-hpc/. [Accessed: 08- Apr- 2015].

[58] My-inner-voice.blogspot.com, 'The memories of a Product Manager: Some thoughts on the differences between HTC and HPC', 2015. [Online]. Available: http://my-inner-voice.blogspot.com/2013/03/some-thoughts-on-differences-between.html. [Accessed: 08- Apr- 2015].

[59] T. Hamada, K. Nitadori, K. Benkrid, Y. Ohno, G. Morimoto, T. Masada, Y. Shibata, K. Oguri and M. Taiji, 'A novel multiple-walk parallel algorithm for the Barnes–Hut treecode on GPUs – towards cost effective, high performance N-body simulation', Comp. Sci. Res. Dev., vol. 24, no. 1-2, pp. 21-31, 2009.

[60] J. Carroll, 'How To: Building Your Own Render Farm - Introduction', Tom's Hardware, 2010. [Online]. Available: http://www.tomshardware.com/reviews/render-farm-node,2340.html. [Accessed: 08- Apr- 2015].

[61] 'Design of SOA-based Grid Computing with Enterprise Service Bus', AISS, vol. 2, no. 1, pp. 71-82, 2010.

[62] Wikipedia, 'Parallel computing', 2015. [Online]. Available: http://en.wikipedia.org/wiki/Parallel_programming. [Accessed: 08- Apr- 2015].

[63] Wikipedia, 'Message Passing Interface', 2015. [Online]. Available: http://en.wikipedia.org/wiki/Message_Passing_Interface. [Accessed: 08- Apr- 2015].

[64] Wikipedia, 'CentOS', 2015. [Online]. Available: http://en.wikipedia.org/wiki/CentOS. [Accessed: 08- Apr- 2015].

[65] Wikipedia, 'GNU Compiler Collection', 2015. [Online]. Available: http://en.wikipedia.org/wiki/GNU_Compiler_Collection. [Accessed: 08- Apr- 2015].

[66]J. Smith and Ravi Nair, 'The architecture of virtual machines', Computer, vol. 38, no. 5, pp. 32-38, 2005.

[67] Wikipedia, 'Network File System', 2015. [Online]. Available: http://en.wikipedia.org/wiki/Network_File_System. [Accessed: 08- Apr- 2015].

[68] Wikipedia, 'Network monitoring', 2015. [Online]. Available: http://en.wikipedia.org/wiki/Network_monitoring. [Accessed: 08- Apr- 2015].

[69] Wikipedia, 'Secure Shell', 2015. [Online]. Available: http://en.wikipedia.org/wiki/Secure_Shell. [Accessed: 08- Apr- 2015].

[70] Wikipedia, 'HTCondor', 2015. [Online]. Available: http://en.wikipedia.org/wiki/HTCondor. [Accessed: 08- Apr- 2015].

[71] Wikipedia, 'Software Framework', 2015. [Online]. Available: http://en.wikipedia.org/wiki/Software_Framework. [Accessed: 08- Apr- 2015].

[72] Wikipedia, 'Algorithmic skeleton', 2015. [Online]. Available: http://en.wikipedia.org/wiki/Algorithmic_skeleton#Muesli. [Accessed: 08- Apr- 2015].

[73] Wikipedia, 'Abstraction (computer science)', 2015. [Online]. Available: http://en.wikipedia.org/wiki/Abstraction_%28computer_science%29. [Accessed: 08- Apr- 2015].

[74] Computing.llnl.gov, 'Introduction to Parallel Computing', 2015. [Online]. Available: https://computing.llnl.gov/tutorials/parallel_comp/#DesignUnderstand. [Accessed: 08- Apr- 2015].

[75] Computing.llnl.gov, 'Introduction to Parallel Computing', 2015. [Online]. Available: https://computing.llnl.gov/tutorials/parallel_comp/#DesignDependencies. [Accessed: 08- Apr- 2015].

[76] T. Hamada, K. Nitadori, K. Benkrid, Y. Ohno, G. Morimoto, T. Masada, Y. Shibata, K. Oguri and M. Taiji, 'A novel multiple-walk parallel algorithm for the Barnes–Hut treecode on GPUs – towards cost effective, high performance N-body simulation', Comp. Sci. Res. Dev., vol. 24, no. 1-2, pp. 21-31, 2009.

[77] Computing.llnl.gov, 'Introduction to Parallel Computing', 2015. [Online]. Available: https://computing.llnl.gov/tutorials/parallel_comp/#DesignSynchronization. [Accessed: 08- Apr- 2015].

[78] Wikipedia, 'Software metric', 2015. [Online]. Available: http://en.wikipedia.org/wiki/Software_metric. [Accessed: 08- Apr- 2015].

[79] Wikipedia, 'Programming complexity', 2015. [Online]. Available: http://en.wikipedia.org/wiki/Programming_complexity. [Accessed: 08- Apr- 2015].

[80] Wikipedia, 'Halstead complexity measures', 2015. [Online]. Available: http://en.wikipedia.org/wiki/Halstead_complexity_measures. [Accessed: 08- Apr- 2015].

[81] Wikipedia, 'Cyclomatic complexity', 2015. [Online]. Available: http://en.wikipedia.org/wiki/Cyclomatic_complexity. [Accessed: 08- Apr- 2015].

[82] M. Monroy, 'Arquitecturas Avanzadas de Software (Notas de Clase)', Cartagena, 2014.

[83] Hpcinnovationcenter.llnl.gov, 'Livermore's HPC Ecosystem | HPC Innovation Center', 2015. [Online]. Available: http://hpcinnovationcenter.llnl.gov/livermores-hpc-ecosystem. [Accessed: 22- May- 2015].

[84] H. Boyle and E. Chang, 'Digital ecosystems: Principles and semantics', National Research Council Canada, 2007.

[85] Rausch, A.; Muller, J.P.; Niebuhr, D.; Herold, S.; Goltz, U., "IT ecosystems: A new paradigm for engineering complex adaptive software systems," Digital Ecosystems Technologies (DEST), 2012 6th IEEE International Conference on , vol., no., pp.1,6, 18-20 June 2012

[86] Digital Ecosystem Convergence between IT, Telecoms, Media and Entertainmnet: Scenarios 2015, 1st ed. 2015.

ANEXOS

Anexo 1: Documento de Arquitectura del Sistema

[Son 50 páginas de extensión, se anexa por separado]

Anexo 2: Archivo de código fuente 'FDDC.cpp'

```
/*
 * FDDC.cpp
 *
 *  Created on: Jan 20, 2015
 *      Author: mpiuser
 */
#include <iostream>
#include <math.h>
#include "mpi.h"
#include "F.h"
#include "FDDC.h"
using namespace std;

FDDC::FDDC(){};

void FDDC::FDDCK2()
{
      int n=3;
      //int n=15;
      int p=n*n;
      int v=7;
      //int v=127;
      int fill=0;
      int processors[n][n];
      //la matriz de profundidades debemos elaborarla manualmente
      int depths[3][3]={
                  {2,1,2},
                  {0,0,0},
                  {2,1,2},
      };
      /*int depths[15][15]={
                  {6,5,6,0,0,0,0,0,0,0,0,0,6,5,6},
                  {0,4,0,3,0,4,0,0,0,4,0,3,0,4,0},
                  {6,5,6,0,0,0,0,0,0,0,0,0,6,5,6},
                  {0,0,0,2,0,0,0,1,0,0,0,2,0,0,0},
                  {0,5,0,0,0,0,0,0,0,0,0,0,6,5,6},
                  {0,4,0,3,0,4,0,0,0,4,0,3,0,4,0},
                  {6,5,6,0,0,0,0,0,0,0,0,0,6,5,6},
```

```
                {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                {6,5,6,0,0,0,0,0,0,0,0,0,6,5,6},
                {0,4,0,3,4,0,0,0,0,4,0,3,0,4,0},
                {6,5,6,0,0,0,0,0,0,0,0,0,6,5,6},
                {0,0,0,2,0,0,0,1,0,0,0,2,0,0,0},
                {0,5,0,0,0,0,0,0,0,0,0,0,6,5,6},
                {0,4,0,3,0,4,0,0,0,4,0,3,0,4,0},
                {6,5,6,0,0,0,0,0,0,0,0,0,6,5,6},
        };*/

        //hojas (nodos destino) de cada procesadro (nodo fuente)
        int leaves[p][2];
        //vector que guarda las pseudo coordenadas del pseudo plano
cartesiano
        int coords[p][2];
        //comunicadores y grupos
        int comms[3][3];

        //order processors
        for(int i=0;i<n;i++)
        {
              for(int j=0;j<n;j++)
              {
                    processors[i][j]=fill;
                    fill++;
              }
        }
        //assign coordinates to processors
        int sig1=-1;
        int sig2=1;
        int k=0;
        for(int i=0;i<n;i++)
        {
              for(int j=0;j<n;j++)
              {
                    if(i==n)
                    {
                          sig1=sig1*-1;
                          sig2=sig2*-1;

                    }
```

```
                coords[k][0]=((n/2)-j)*sig1;
                coords[k][1]=((n/2)-i)*sig2;

                //cout<<"proc = "<<processors[i][j]<<" tiene
coords("<<coords[k][0]<<", "<<coords[k][1]<<") con depth
="<<depths[i][j]<<"\n";
                k++;
          }
     }
     //Generar hijos de procesadores
     int xi, yi, logVal=log2((v+1)/2), d, x1, x2, y1, y2, p1, p2;

     for(int i=0;i<n;i++)
     {
          for(int j=0;j<n;j++)
          {    //si el depth es mayor que 0 o es la raiz,
entonces tiene hijos
                if((depths[i][j]!=0 && depths[i][j]!=logVal) ||
(i==n/2 && j==n/2))
                {
                     xi=coords[processors[i][j]][0];
                     yi=coords[processors[i][j]][1];
                     d=depths[i][j];
                     //determinar ambos hijos dependiendo de si
logVal y d son pares, impares o ambos
                     if((logVal%2==0 && d%2==0) || (logVal%2!=0 &&
d%2!=0))
                     {
                          x1=xi;

     y1=yi+pow(2.0,(1.0/2.0)*(log2(((double)v+1.0)/2.0)-(double)d-
2.0));
                          x2=xi;
                          y2=yi-
pow(2.0,(1.0/2.0)*(log2(((double)v+1.0)/2.0)-(double)d-2.0));
                     }
                     else
                     {
```

```
      x1=xi+pow(2.0,(1.0/2.0)*(log2(((double)v+1.0)/2.0)-(double)d-
1.0));;
                              y1=yi;
                              x2=xi-
pow(2.0,(1.0/2.0)*(log2(((double)v+1.0)/2.0)-(double)d-1.0));
                              y2=yi;
                        }
                        //buscar los procesos hijos
                        for(int k=0;k<p;k++)
                        {
                              if(coords[k][0]==x1 &&
coords[k][1]==y1)
                              {     p1=k;
                              }
                              if(coords[k][0]==x2 &&
coords[k][1]==y2)
                              {     p2=k;
                              }
                        }
                        //Cada procesador ya conoce sus hijos
                        leaves[processors[i][j]][0]=p1;
                        leaves[processors[i][j]][1]=p2;
                  }
                  else
                  {
                        //Los procesadores sin hijos se les asigna un
valor hijo comod[in
                        leaves[processors[i][j]][0]=999;
                        leaves[processors[i][j]][1]=999;
                  }

                  //cout<<"Proc "<<processors[i][j]<<", ("<<xi<<",
"<<yi<<") tiene de hijos a proc "<<leaves[processors[i][j]][0]<<",
("<<x1<<", "<<y1<<") y proc "<<leaves[processors[i][j]][1]<<",
("<<x2<<", "<<y2<<")\n";
            }
      }

      //necesitamos construir los comunicadores y grupos con los
procesadores correspondientes
```

```
	k=0;
	for(int i=0;i<n;i++)
	{
		for(int j=0;j<n;j++)
		{
			if(leaves[processors[i][j]][0]<p &&
leaves[processors[i][j]][1]<p)
			{
				//En la primera columna quedan los padres
				comms[k][0]=processors[i][j];
				comms[k][1]=leaves[processors[i][j]][0];
				comms[k][2]=leaves[processors[i][j]][1];

				//cout<<"El comunicador "<<k<<" tiene
procesos "<<comms[k][0]<<", "<<comms[k][1]<<",
"<<comms[k][2]<<"\n";

				k++;
			}
		}
	}


	/* Inicializamos varibles para calcular arbol de procesos en
una "grilla con coordenadas cartesianas"
	 * inbufVal guarda los datos que son recibidos por paso de
mensajes
	 * tag es un entero que identifica una operacion send y
receive conjunta
	 *	newRank el rango de los procesos cuando se asignan al
nuevo comunicador
	 *	source y dest guarda el rango de los pro. fuente destino
y fuente en operaci[on conjunta send receive
	 *	color es un entero que clasifica los proc. en diferentes
comunicadores
	 *	nameLen guarda el tamano de la cadena que almacena el
nombre de los hosts del cluster
	 *	key es un entero que MPI_Split utiliza para ordenar los
procesos dentro del comunicador nuevo
	 */
```

```
      int numtasks, rank, source, dest, outbufVal=20,
i,j,tag,color,nameLen,
         newRank,maxNbrs=3,numProc=9,key,initialized,finalized;
      char const *colorName[] = { "BLACK", "BLUE", "WHITE" };
      char processorName[MPI_MAX_PROCESSOR_NAME];

      //vector de 13 terminos a los que se aplicara una funcion
      //el manejo de los vectores es vital para el exito del
aplicativo
      //problemas de memoria puede causar abortaje de procesos
      //los modificadores * pueden ser la causa
      //http://www2.warwick.ac.uk/fac/sci/dcs/people/research/csrcb
c/teaching/hpcseminars/seminars07/mpierrors/
      prepareF();
      double terms[12];
      for(i=0;i<numElementsInputArray;i++)
      {
            terms[i]=inputArray[i];
      }
      int numTerm=numElementsInputArray,termCount=numTerm/6;
      double
localArray1[numTerm/2],localArray2[numTerm/2],la[numTerm/2],localS
um[termCount],gatherArray[n];
      double sum,inbufVal,inbufVal1,inbufVal2;


      //separamos el array principal en dos, cada parte se envia en
un hijo de la raiz, y este luego procede a reducir con sus hojas
      //si el vector es impar queda el ultimo elemento por fuera
por ahora
      for(i=0;i<numTerm/2;i++)
      {
            localArray1[i]=terms[i];
      }
      j=numTerm/2;
      for(i=0;i<numTerm/2;i++)
      {
            localArray2[i]=terms[j];
            j++;
      }
```

```
      MPI_Initialized(&initialized);

      if(!initialized)
      {
            MPI_Init(NULL,NULL);
      }

      /* el objeto MPI_Request es multiproposito, manejadores de
pasos de mensajes
       * MPI_Stats es un objeto multiproposito que guarda el estado
de pasos de mensajes
       * MPI_COMM_WORLD es el comunicador que se empleara
       */
      MPI_Request reqs[6];
      MPI_Status status;

      //creamos los comunicadores que gestionan intercomunicacion
en el arbol
      MPI_Comm commLocal,comm2,comm3;

      /*Iniciamos entorno MPI y un comunicador
      * A partir de MPI_Init todos los procesadores ejecutan
      * individualmente el codigo a continuacion
      * hasta la instruccion MPI_Finaliza
      *
      * Buscando en internet, algunas fuentes aseguran que Init y
Finalize
      * no garantizan ejecucion unica de codigo no-mpi antes del
bloque mpi
      * http://stackoverflow.com/questions/9084449/mpi-serial-main-
function
      */
      //MPI_Init(&argc,&argv);
      MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
      //se asigna un identificador a cada procesador
      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
      //tomamos el host del procesador
      MPI_Get_processor_name(processorName, &nameLen);

      if(numtasks == numProc || rank!=3 || rank!=5)
```

```
     {     /*validamos que exista el numero de procesadores
deseados*/


           //imprimimos informacion de cada procesador en su
comunicador global
           printf("Hello world! I'm  rank %d of %d on %s\n",
rank,numtasks,processorName);

           //construimos intra-comunicador de cada grupo
           //primero buscamos el "color" de cada procesador
           //con los colores distinguimos los comunicadores
           for(j=0;j<maxNbrs;j++)
           {
                 if((j!=ceil(maxNbrs/2)) && (comms[j][0]==rank ||
comms[j][1]==rank || comms[j][2]==rank))
                 {
                       /*cada terna del vector comm debe ser un
comunicador, a excepcion del nodo raiz que
                       quedaria solito en su comunicador local
                       no queremos que ningun procesador a parte de
la raiz quede en el comunicador principal*/
                       color=j;
                 }
                 else if(rank==p/2)
                 {
                       //sabemos que el nodo raiz debe ir en un
comunicador principal en solitario
                       color=1;
                 }

                 //organizamos los comunicadores por orden
dispuesto previamente en el vector comms
                 for(k=0;k<maxNbrs;k++)
                 {
                       if(comms[j][k]==rank)
                       {
                             key=k;
                       }
                 }
           }
```

```
          /*creamos tres communicadores locals. Existiran 3
commLocal
          los lideres locales en caso de usar 9 procesadores seria
los proc
          1, 4 y 7. Estos en su contexto local se
          re-identificaran con id 0*/
          MPI_Comm_split(MPI_COMM_WORLD,color,key, &commLocal);
          //los procesadores en su comm local adquieren nuevos ids
          MPI_Comm_rank(commLocal, &newRank);

          printf("%d: I'm  rank %d of %d in the %s
context\n",rank,newRank,maxNbrs,colorName[color]);

          //separamos los grupos en intercomunicadores
          if (color==0)
           {
               //creamos un intercomunicador entre un lider en
contexto local con su padre que es el nodo raiz en el contexto
global
               source=0;
               dest=4;
               tag=99;
               MPI_Intercomm_create( commLocal, source,
MPI_COMM_WORLD, dest,tag, &comm2);

               if(newRank==0)
               {
                    //recibimos primero el mensaje
                    source=newRank,tag=1;
                    MPI_Recv(&la, numTerm/2,
MPI_DOUBLE,source,tag,comm2,&status);

                    //Hacemos Scatter de la parte de datos
recibidos

     MPI_Scatter(la,termCount,MPI_DOUBLE,localSum,termCount,MPI_DO
UBLE,source,commLocal);

                    sum=functionPostScatter(termCount,localSum);
                    //Hacemos Gather de los datos procesados
```

```
     MPI_Gather(&sum,1,MPI_DOUBLE,gatherArray,1,MPI_DOUBLE,source,
commLocal);

                        sum=functionMaster(newRank,n,gatherArray);
                        //enviamos la variable acumuladora de un
lider local a su padre en su contexto local
                        dest=newRank,tag=11;
                        MPI_Send(&sum, 1, MPI_DOUBLE,dest, tag,
comm2);
                }
                else
                {
                        //Hacemos Scatter de la parte de datos
recibidos

     MPI_Scatter(la,termCount,MPI_DOUBLE,localSum,termCount,MPI_DO
UBLE,source,commLocal);

                        //Hacemos Gather de los datos procesados
                        sum=functionPostScatter(termCount,localSum);

     MPI_Gather(&sum,1,MPI_DOUBLE,gatherArray,1,MPI_DOUBLE,source,
commLocal);
                }
                //liberamos el intercomunicador despues de la
comunicacion
                MPI_Comm_free(&comm2);
            }
            else if (color==1) //Este comm local es el del nivel
superior y recibe datos de los demas comms locales
            {
                 /* el procesador raiz en su contexto local se
comunica con sus hijos en el contexto global*/

                source=0,dest=1,tag=99;
                MPI_Intercomm_create(commLocal, source,
MPI_COMM_WORLD, dest,tag, &comm2);
                dest=7,tag=66;
                MPI_Intercomm_create(commLocal, source,
MPI_COMM_WORLD, dest,tag, &comm3);
```

```
                if(newRank==0)
                {
                      //enviamos los mensajes vector primero
                      dest=newRank,tag=1;
                      MPI_Send(&localArray1, numTerm/2,
MPI_DOUBLE,dest, tag, comm2);
                      tag=2;
                      MPI_Send(&localArray2, numTerm/2, MPI_DOUBLE,
dest, tag, comm3);

                      /* el procesador raiz recibe datos de sus
hijos, que son lideres
                       * en sus respectivos comunicadores */
                      source=newRank,tag=11;
                      MPI_Recv(&inbufVal1, 1,
MPI_DOUBLE,source,tag,comm2,&status);
                      tag=22;
                      MPI_Recv(&inbufVal2, 1,
MPI_DOUBLE,source,tag,comm3,&status);

                      //sumamos lo recibido y calculamos la media
con la funcion final
                      functionFinal(newRank,inbufVal1,inbufVal2);
                }

                //liberamos los intercomunicadores una vez cesamos
el paso de mensajes
                MPI_Comm_free(&comm2);
                MPI_Comm_free(&comm3);
            }
            else if (color == 2)
            {
                 //creamos un intercomunicador entre un lider en
contexto local con su padre que es el nodo raiz en el contexto
global
                source=0;
                dest=4;
                tag=66;
                MPI_Intercomm_create(commLocal, source,
MPI_COMM_WORLD,dest,tag, &comm3);
```

```
                if(newRank==0)
                {     source=newRank;
                      tag=2;
                      MPI_Recv(&la, numTerm/2,
MPI_DOUBLE,source,tag,comm3,&status);

                      //Hacemos Scatter de la parte de datos
recibidos

     MPI_Scatter(la,termCount,MPI_DOUBLE,localSum,termCount,MPI_DO
UBLE,source,commLocal);

                      sum=functionPostScatter(termCount,localSum);
                      //Hacemos Gather de los datos procesados

     MPI_Gather(&sum,1,MPI_DOUBLE,gatherArray,1,MPI_DOUBLE,source,
commLocal);

                      sum=functionMaster(newRank,n,gatherArray);
                      //enviamos la variable acumuladora de un
lider local a su padre en su contexto local
                      dest=newRank;
                      tag=22;
                      MPI_Send(&sum, 1, MPI_DOUBLE, dest, tag,
comm3);

                }
                else
                {
                      //Hacemos Scatter de la parte de datos
recibidos

     MPI_Scatter(la,termCount,MPI_DOUBLE,localSum,termCount,MPI_DO
UBLE,source,commLocal);

                      sum=functionPostScatter(termCount,localSum);
                      //Hacemos Gather de los datos procesados

     MPI_Gather(&sum,1,MPI_DOUBLE,gatherArray,1,MPI_DOUBLE,source,
commLocal);
                }
```

```
                //liberamos el intercomunicador despues del paso
de mensajes
                MPI_Comm_free(&comm3);
             }
           MPI_Comm_free(&commLocal);

           MPI_Finalized(&finalized);

           if(!finalized)
           {
                MPI_Finalize();
           }
      }
      else
      {
           //no se encuentra el numero de procesos deseado o el
procesador no hace parte del arbol
           printf("Must specify %d processors.
Terminating.\n",numProc);
      }

}
```

Anexo 3: Archivo de código fuente ‘F.h’

```
/*
 * F.cpp
 *
 *  Created on: Jan 20, 2015
 *      Author: mpiuser
 */
#include <iostream>
#include <math.h>
#include "mpi.h"
#include "GeometricalManager.h"
using namespace std;


int numElementsInputArray=12;
double inputArray[12];

void prepareF()
{
     inputArray[0]=50.0;inputArray[1]=50.0;inputArray[2]=50.0;inpu
tArray[3]=50.0;inputArray[4]=50.0;inputArray[5]=50.0;inputArray[6]
=50.0;inputArray[7]=50.0;inputArray[8]=50.0;inputArray[9]=50.0;inp
utArray[10]=50.0;inputArray[11]=50.0;
}


double function(int rounds)
{
}

double functionMaster(int rank, int n, double gatherArray[])
{

void functionFinal(int rank,double val1, double val2)
{
}

double functionPostScatter(int termCount, double localSum[])
{
```

```
}
```

Anexo 4: Archivo de código fuente ‘main.cpp’

```
/*
 * Split.cpp
 *
 *  Created on: Sep 14, 2014
 *      Author: mpiuser
 */

#include "FDDC.h"
using namespace std;

int main(int argc, char *argv[])
{
      FDDC fddc;
      fddc.FDDCK2();

      return 0;
}
```

Anexo 5: Archivo de código fuente ‘main.cpp’ (versión MPI Puro)

```
/*
 * main.cpp
 *
 *  Created on: Jan 10, 2014
 *      Author: mpiuser
 */
#include "mpi.h"
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <math.h>
#include "GeometricalManager.h"
using namespace std;

int main(int argc, char *argv[])
{
      int id;
      int numTasks;

      double finalPi=0;
      double averagePi=0;
      double localPi=0;
      double sumPi=0;
      double rounds=100;
      int numPoints=10000;
      int circleCount=0;
      double r;
      Circle circle;
      Rectangle rectangle;
      GeometricalManager gm;

      MPI_Init(&argc,&argv);
      MPI_Comm_size(MPI_COMM_WORLD, &numTasks);
      MPI_Comm_rank(MPI_COMM_WORLD, &id);

      r=1;
      circle.setRadius(r);
      rectangle.setLength(2*r);
```

```
	cout<<"MPI task "<<id<<" has started\n";


	for(int i=0;i<rounds;i++)
	{
		localPi=gm.computePi(r,numPoints,circleCount,rectangle);


	MPI_Reduce(&localPi,&sumPi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WO
RLD);

			if(id==0)
			{
				finalPi=sumPi/numTasks;
				averagePi=((averagePi * i) + finalPi)/(i + 1);
			}
	}

	if(id==0)
	{
			circle.setPi(averagePi);
			circle.setArea();
			cout<<"Number of points used is "<<numPoints<<"\n";
			cout<<"Circle's radius is "<<circle.getRadius()<<"\n";
			cout<<"Average value of Pi is "<<circle.getPi()<<"\n";
			cout<<"The average value of the circle's area is
"<<circle.getArea()<<"\n";
	}

	MPI_Finalize();
	return 0;

}
```

Anexo 6: Archivo de código fuente ‘F.h’ (versión framework)

```
/*
 * F.cpp
 *
 *  Created on: Jan 20, 2015
 *      Author: mpiuser
 */
#include <iostream>
#include <math.h>
#include "mpi.h"
#include "GeometricalManager.h"
using namespace std;


int numTasks=3;//;El arbol tiene 9 procesadores, de los cuales 6
ejecutan el algoritmo, 3 procesadores por rama del arbol
double averagePi=0.0;
double rounds=100;
int numPoints=10000;
int circleCount=0;
double r=1;
Circle circle;
Rectangle rectangle;
GeometricalManager gm;
//estos dos elementos son esenciales en el algoritmo
int numElementsInputArray=12;
double inputArray[12];

void prepareF()
{
      circle.setRadius(r);
      rectangle.setLength(2*r);
      inputArray[0]=10.0;inputArray[1]=10.0;inputArray[2]=10.0;inpu
tArray[3]=10.0;inputArray[4]=10.0;inputArray[5]=10.0;inputArray[6]
=10.0;inputArray[7]=10.0;inputArray[8]=10.0;inputArray[9]=10.0;inp
utArray[10]=10.0;inputArray[11]=10.0;
}
```

```
double function(int rounds)
{
      double localPi=0;
      for(int i=0;i<rounds;i++)
      {

      localPi+=gm.computePi(r,numPoints,circleCount,rectangle);
      }
      //cout<<"localPi: "<<localPi/rounds<<"\n";
      return localPi/rounds;

}

double functionMaster(int rank, int n, double gatherArray[])
{
      double sum=0;
      for(int i=0;i<n;i++)
      {
            sum+=gatherArray[i];
      }
      if(rank==0)
      {
            averagePi=sum/numTasks;
      }
      //cout<<"averagePi: "<<averagePi<<"\n";
      return averagePi;
}

void functionFinal(int rank,double val1, double val2)
{

      averagePi=(val1+val2)/2;
      if(rank==0)
      {
            circle.setPi(averagePi);
            circle.setArea();
            cout<<"Circle's radius is "<<circle.getRadius()<<"\n";
            cout<<"Average value of Pi is "<<circle.getPi()<<"\n";
            cout<<"The average value of the circle's area is
"<<circle.getArea()<<"\n";
      }
```

```
}

double functionPostScatter(int termCount, double localSum[])
{
      double sum=0;
      for(int i=0;i<termCount;i++)
      {
            sum+=function(localSum[i]);
      }
      sum=sum/termCount;
      return sum;
}
```

Anexo 7: Archivo de código fuente 'test.cpp' (versión serial/secuencial)

```
/*
 * main.cpp
 *
 *  Created on: Jan 10, 2014
 *      Author: mpiuser
 */
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <math.h>
#include "GeometricalManager.h"
using namespace std;

int main(int argc, char *argv[])
{
      int id;
      int numTasks;

      double averagePi=0;
      double localPi=0;
      double rounds=100;
      int numPoints=10000;
      int circleCount=0;
      double r;
      Circle circle;
      Rectangle rectangle;
      GeometricalManager gm;

      r=1;
      circle.setRadius(r);
      rectangle.setLength(2*r);

      for(int i=0;i<rounds;i++)
      {

      localPi+=gm.computePi(r,numPoints,circleCount,rectangle);
      }

      averagePi=localPi/rounds;

      circle.setPi(averagePi);
      circle.setArea();
      cout<<"Number of points used is "<<numPoints<<"\n";
      cout<<"Circle's radius is "<<circle.getRadius()<<"\n";
```

```
      cout<<"Average value of Pi is "<<circle.getPi()<<"\n";
      cout<<"The average value of the circle's area is
"<<circle.getArea()<<"\n";

      return 0;

}
```