DISEÑO DE PROTOTIPO DE GUI PARA MONITOREO DE TRABAJOS DEL CLUSTER DE PROCESAMIENTO DE LA UTB

ALVARO MANUEL GARCIA CABRERA
ROIMAN CABEZA BARRIOS

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
FACULTAD DE INGENIERÍA
CARTAGENA DE INDIAS
2013

DISEÑO DE PROTOTIPO DE GUI PARA MONITOREO DE TRABAJOS DEL CLUSTER DE PROCESAMIENTO DE LA UTB

ALVARO MANUEL GARCIA CABRERA
ROIMAN CABEZA BARRIOS

Tesis Para Optar Al Título De Ingeniero de Sistemas

EDWIN ALEXANDER PUERTA DEL CASTILLO
Magister en Ingeniería
Director

UNIVERSIDAD TECNOLOGICA DE BOLÍVAR
FACULTAD DE INGENIERÍA
CARTAGENA DE INDIAS
2013

CONTENIDO

LISTA DE FIGURAS

1. INTRODUCCIÓN

1.1. Planteamiento del Problema

Actualmente, la difusión de la información con el uso del Internet ha crecido increíblemente y es notable la utilidad y los beneficios obtenidos a partir de esto. Por otro lado, la cantidad de implementaciones, a través de la misma Internet, y el almacenamiento de grandes cantidades de datos han hecho de los recursos computacionales una necesidad.

Las necesidades de las organizaciones para el procesamiento de datos son cada vez mayores, debido a esto se ha luchado por conseguir la integración necesaria entre sistemas de diferentes organizaciones, o, aún entre sistemas dentro de una misma organización, sin embargo, dada la diversidad de tecnologías los desarrolladores se ven en la necesidad de construir sobre diversidad de aplicaciones y para diferentes entornos. La comunidad profesional de la información ha estado fascinada con el potencial de unir diferentes ordenadores obteniendo de esta forma un solo ordenador virtual que permita el procesamiento masivo de datos enfocándose en los recursos.
Algunos de los inconvenientes asociados a las soluciones tradicionales y los sistemas basados en la centralización de los datos por medio de un servidor son la falta de escalabilidad, mantenimiento caro, además, una vez adquirido equipo este podría estar mucho tiempo desaprovechado.

Es necesario resaltar que en la lucha por la integración de recursos se han desarrollado algunas soluciones como lo son las arquitecturas clúster y arquitecturas de intranet, logrando cierto acercamiento sin llegar a constituirse en la solución optima debido por ejemplo a la dificultad de mantenimiento, escalabilidad limitada, falta de amortización y distribución de la carga entre recursos cuando estos están desaprovechados, entre otros.

Grid Computing surge como la tecnología que permite la integración de diferentes recursos computacionales para conformar una sola unidad. Nosotros, de manera muy general, la definimos como un servicio que comparte el poder de una computadora y la capacidad de almacenamiento de datos a través del internet. Permite que procesadores, discos, redes, dispositivos y maquinas virtuales dispersos a través de la web interactúen transparentemente.

Sin embargo, hoy por hoy no existen aplicaciones que permitan manejar una grilla computacional de forma sencilla, intuitiva y amigable. El usuario final no debería verse ni siquiera en la necesidad de conocer el protocolo FTP para poder subir un archivo; es necesario el desarrollo de software que permita que este proceso sea totalmente transparente.

1.2. Objetivos

1.2.1. Objetivo General

Definir, explicar y presentar los conceptos y ventajas de Grid Computing mediante el diseño de un prototipo que facilite la transición Usuario – Grilla en la UTB y que a su vez nos permita demostrar el poder que brinda este servicio.

1.2.2. Objetivos Específicos

- Socializar el estado del arte de la computación en la grilla (Grid Computing), para la comunidad académica e investigativa de la Universidad Tecnológica de Bolívar.
- Realizar una exhaustiva revisión bibliográfica referente a la computación Orientada a servicios.
- Apropiarse del conocimiento obtenido en el cumplimiento de los objetivos anteriores.
- Diseño de un prototipo de UI (Interface de usuario) para envío, monitoreo y ejecución de trabajos para el clúster de procesamiento de alto rendimiento de la UTB.

- Apropiarse de la teoría y el uso de herramientas tecnológicas como BPEL, WSDL, SOAP y XML para poder diseñar el prototipo de UI para el clúster de procesamiento de alto rendimiento de la UTB.

1.3. Metodología De Investigación

Revisar y consultar referentes bibliográficas, revistas especializadas, artículos científicos acerca de grid computing, manejo de procesos de clusters mediante Web Services, computación distribuida, arquitecturas orientadas a servicios, XML y Web Services, que permitan la construcción del estado de arte, y constituyan las bases conceptuales de este trabajo.

Posteriormente analizar y comprender los conceptos, orígenes, influencias, metas y principios de Grid Computing, así como la evolución que ha tenido hasta nuestros días, para la apropiación del conocimiento.

Luego, diseñar un prototipo de interface de usuario que brinde un manejo fácil, rápido y amigable al usuario que a su vez permitirá explotar plena y totalmente el poder, las ventajas y los beneficios ofrecidos por la computación grid.

2. PROBLEMÁTICA DE LA CONSTRUCION DE APLICACIONES PARA GRID COMPUTING

En todo el mundo, los laboratorios científicos elaboran y usan programas que los ayudan a lograr avances en sus investigaciones de un modo que no podrían lograr si no fuese por estos.

La gran desventaja de esto es que surgía una nueva necesidad, la de obtener nuevos y más poderosos computadores para ejecutar estos programas que en su mayoría eran pesados en cuanto a tiempo de procesamiento se refiere.

Además, estos procesos generalmente hacían uso de grandes cantidades de datos y a su vez generaban altos volúmenes de datos procesados.

Se necesitaba una solución eficiente para este problema que preocupaba a los investigadores, y ésta debería ser eficiente en el uso de sus limitados recursos.

Para solucionar este problema existían diferentes alternativas, cada una con sus ventajas y desventajas:

- **Supercomputadoras**

Las supercomputadoras, muchos procesadores (siempre potencias de 2) en una arquitectura paralela que brindan un enorme poder de cómputo en los cuales se ejecutan procesos separados en cada uno para realizar tareas muy pesadas en un tiempo exponencialmente menor que si se usara una arquitectura tradicional.

Ventajas

- Alto rendimiento
- Poco espacio

Desventajas

- Altos costos
- Escalabilidad limitada

- **Clusters**

Los clusters, muchos computadores trabajando en conjunto para lograr determinadas tareas. Un concepto que nació a partir de las supercomputadoras, ofreciendo alto rendimiento para cálculos muy pesados, pero poco a poco evolucionó para brindar otras opciones como alta disponibilidad, almacenamiento, balance de carga, entre otras.

Ventajas

- Bajos costos
- Alto nivel de escalabilidad

Desventajas

- Mucho espacio ocupado
- Altos consumos de energía eléctrica
- Necesitan grandes sistemas de enfriamiento

- **Peer-to-Peer**

Peer-to-Peer, comunicación entre pares, nació como una forma de compartir archivos, pero pronto se descubrió su potencial para aprovechar el poder de cómputo a nivel mundial.

- **Surgimiento de Grid**

La palabra grid en inglés significa malla y hace referencia a la red eléctrica. Para obtener electricidad, simplemente debemos conectarnos a cualquier punto de la malla eléctrica (power grid) sin preocuparnos por las plantas generadoras que interconectadas brindan esta omnipresencia del servicio, es decir, no sabemos de dónde proviene la electricidad que usamos, simplemente la aprovechamos.[1]
Este fue el concepto que adaptaron Carl Kesselman e Ian Foster en su libro The Grid: Blueprint for a new computing infrastructure publicado en 1998.

La idea que presentaron en su publicación era la de crear una red mundial de laboratorios proveedores de poder de cómputo y capacidad de almacenamiento, así como lo hacen las plantas eléctricas para proveer la electricidad, y que el acceso a ellos fuese fácil, permitiendo a todos aprovechar la capacidad de este gran conjunto heterogéneo de sistemas sin necesidad de preocuparse por la interconexión ya que (como enchufar un electrodoméstico) no importaría que o cuales nodos de esta red me proveen el servicio.

Cada uno de los conceptos antes enunciados entroncaba una arquitectura asociada al mismo que iba haciendo desaparecer al anterior. La necesidad de una mayor capacidad de computación para resolver los desafíos científicos y técnicos que nos encontramos en los problemas reales del día a día (estudio del genoma, necesidad de hallar vacunas contra el sida, búsqueda de inteligencia en el espacio exterior, simulaciones de riesgos en evoluciones de mercados financieros…) ha provocado que los problemas sean abordados desde la perspectiva de la utilización y colaboración entre distintos recursos heterogéneos y dispersos geográficamente. Se hace necesario utilizar y maximizar los recursos existentes empleando toda la capacidad productiva que pueden ofrecer.

Aunque la idea de aprovechar recursos de una parte de la red informática que no se está utilizando es básicamente antigua, aparece en un programa llamado warm, que se utilizo en las primeras redes Ethernet del laboratorio de PARC (Palo Alto Research Center), de Xerox, mayoritariamente se ha entendido que podemos centrar el origen del Grid en el proyecto SETI@home, que sucedió a otro de la NASA denominado SETI, siglas inglesas de Search for Extra-Terrestrial Inteligence, en el que se buscaba señales de inteligencia extraterrestre por medio de las observaciones de potentes radiotelescopios. El proyecto SETI@home continuaba lo iniciado por la NASA; pero lo que hizo que se considerase el primer Grid fue la utilización de un software que se podía descargar de Internet con el objetivo de utilizar ordenadores personales de los internautas que se bajaban el programa con la intención de aumentar la capacidad de procesamiento necesario

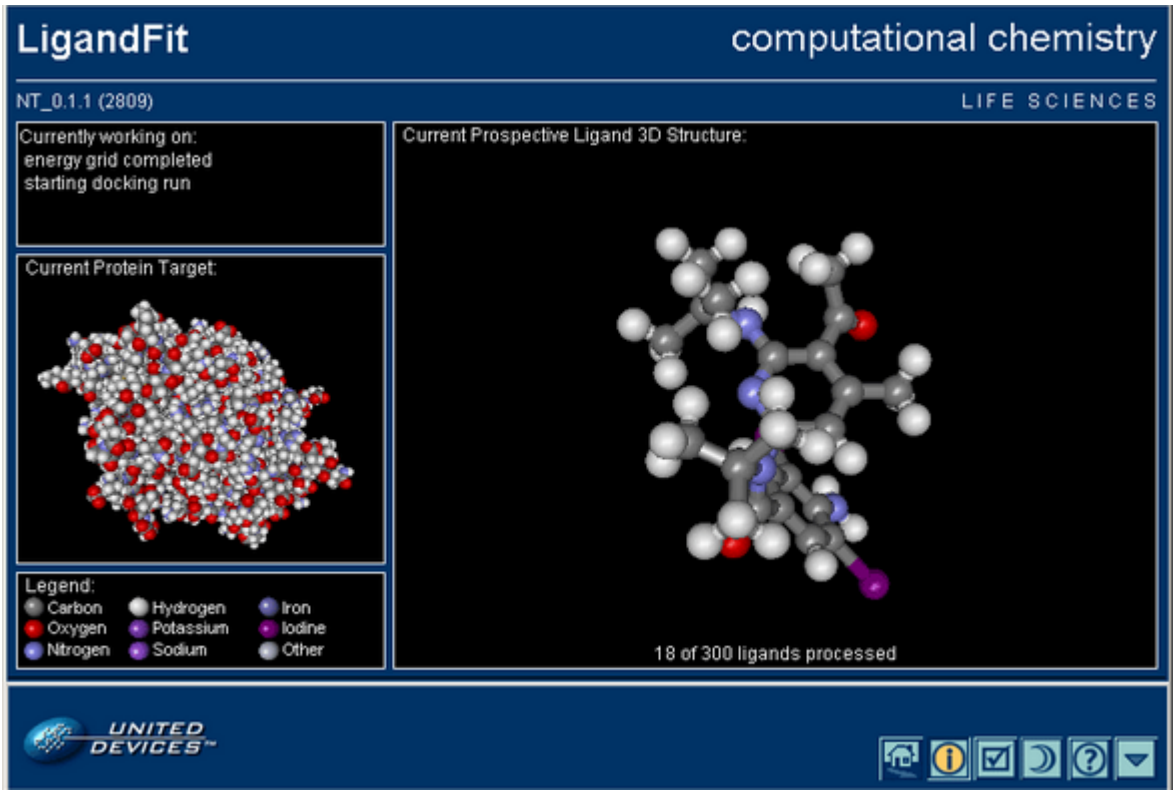
para analizar las señales proporcionadas por el radiotelescopio de Arecibo (Puerto Rico).

Como resultado del proyecto más de un millón y medio de personas han permitido que sus ordenadores personales colaboren con el proyecto. Los resultados de estos colaboradores anónimos de más de 200 países han sido espectaculares: Una medida de 10 trillones de operaciones por segundo y el equivalente a más de 150.000 años de tiempo de computación.

Las instituciones y organismos más interesadas en el desarrollo del grid son, principalmente, las que comparten un objetivo común y que, para poder alcanzarlo, lo más efectivo es compartir sus recursos: Gobiernos y organizaciones internacionales (respuesta a desastres, planificación urbana, etc.); sanidad (análisis rápido de imágenes médicas complejas, etc.); educación (creación de aulas virtuales, teleconferencias, etc.), empresas y grandes corporaciones (cálculos complejos, reuniones virtuales, etc.).

Los beneficios del grid, gracias a la integración de recursos distribuidos, están teniendo repercusión en muchísimos campos, de entre los que cabe destacar: medicina (imágenes, diagnosis y tratamiento), ingeniería genética y biotecnología (estudios en genómica y proteómica), nanotecnología (diseño de nuevos materiales a escala molecular), ingeniería (diseño, simulación, análisis de fallos y acceso remoto a instrumentos de control), y recursos naturales y medio ambiente (previsión meteorológica, observación del planeta, modelos y predicción de sistemas complejos).

Figura 1. Aplicación de Grid Computing en la bioinformática.



Fuente: Ramón Jesús Millán Tejedor, Manual Formativo n° 43., http://www.ramonmillan.com/tutoriales/gridcomputing.php

En la actualidad existen diferentes proyectos que resuelven problemas utilizando la grid computing, por lo que se desarrollarán aquellos que específicamente realizan esta clase de computación:

a) BOINC: [2]

(Berkeley Open Infrastructure for Network Computing - Infraestructura Abierta de Berkeley para la Computación en Red). Es una infraestructura para la computación distribuida, desarrollada originalmente para el proyecto SETI@home, pero que actualmente se utiliza para diversos campos como física, medicina nuclear, climatología, etc. La intención de este proyecto es obtener una capacidad

de computación enorme utilizando computadores personales alrededor del mundo. Los proyectos en los que trabaja este software tienen un denominador común, y es que requieren una gran capacidad de cálculo.
BOINC ha sido desarrollado por un equipo ubicado en Space Sciences Laboratory (Laboratorio de Ciencias Espaciales) en la Universidad de California en Berkeley, liderado por David P. Anderson, que también lidera SETI@home. Como una plataforma "quasi-supercomputador", tiene alrededor de 527,880 computadores activos (hosts) alrededor del mundo. [2]
La plataforma puede correr bajo varios sistemas operativos, incluyendo Microsoft Windows y varios sistemas Unix-like incluyendo Mac OS X, Linux y FreeBSD.
BOINC es software libre y disponible bajo la licencia GNU LGPL (Licencia Pública General Reducida de GNU).
Además, esta plataforma puede utilizarse para realizar computación en red dentro de una misma organización, por ejemplo para realizar cálculos propios que la misma empresa necesita dentro de su negocio. Las características mas importantes de BOINC se muestran a continuación:

- Autonomía de proyectos. Varios proyectos pueden utilizar BOINC para realizar el procesamiento. Cada uno de los cuales se maneja como un proyecto independiente, el cual posee su propio servidor y base de datos.

- Flexibilidad del voluntariado. Cada uno de los voluntarios pueden participar en múltiples proyectos, dependiendo de los intereses de cada uno, y además, decidir la cantidad de recursos donados a cada uno de los proyectos en los cuales participa.

- Multiplataforma. El BOINC cliente es encontrado para múltiples plataformas, como Windows, Linux y MacOS.

BOINC ha sido diseñado para soportar aplicaciones que necesiten grandes cantidades de recursos de computo, siendo el único requerimiento que esta

aplicación sea divisible en gran cantidad de tareas pequeñas que puedan ser realizadas, cada una, de manera independiente.

b) SETI@HOME: [2]

(Search ExtraTerrestrial Intelligence - Búsqueda de Inteligencia Extraterrestre).
Es el proyecto por el cual BOINC fue creado, para salvaguardar algunos aspectos de seguridad y de control de los resultados que eran enviados por los usuarios pertenecientes a la red.
SETI es una red científica liderada por el David Anderson, que también lidera el equipo encargado de BOINC, cuya meta es la detección de vida inteligente fuera de nuestro planeta. [2] El proyecto posee radio telescopios que monitorean el espacio captando ondas de radio de banda estrecha, las cuales se saben, no ocurren de forma natural, por lo que su detección seria una evidencia de tecnología extraterrestre.
Las señales de los radiotelescopios consisten en ruidos provenientes de fuentes celestiales y señales producidas por el hombre. Los proyectos de radio SETI se encargan del análisis de estas señales, las cuales se procesan digitalmente, por lo que una potencia mayor de cálculo permite que la búsqueda cubra un mayor rango de frecuencias con una mayor sensibilidad.
Inicialmente estos cálculos se realizaban mediante computadoras en los mismos radiotelescopios, las cuales se encargaban de procesar la mayor cantidad de información. En 1995, David Geyde, propuso que radio SETI tuviera una supercomputadora virtual conectada a través de Internet. El proyecto SETI@Home fue lanzado en mayo de 1999.
El proyecto SETI@Home posee diversos recursos computacionales, como servidores de base de datos para información de usuarios, listas, unidades de trabajo, resultados, información procesada; servidores web para el hosting de la web del proyecto. Cada uno de estos servicios está dividido en 10 servidores, la mayoría de los cuales utilizan tecnología Intel.

Si bien el proyecto no ha detectado ninguna señal de ETI (Inteligencia Extraterrestre), ha identificado varios objetivos candidatos (posiciones celestes), donde el pico de intensidad no es fácil de explicar cómo puntos de ruido, para su posterior análisis. El astrónomo Seth Shostak ha declarado en 2004 que espera para obtener una señal concluyente y prueba de contacto alienígena entre 2020 y 2025, basado en la ecuación de Drake. Esto implica que un esfuerzo prolongado puede beneficiar a SETI@home, a pesar de sus (actuales) once años de ejecución sin éxito en la detección de ETI.

Aunque el proyecto no ha alcanzado el objetivo de encontrar inteligencia extraterrestre, ha demostrado a la comunidad científica que proyectos de computación distribuida que utiliza ordenadores conectados a Internet puede tener éxito como una herramienta de análisis de viabilidad, e incluso ganarle a los más grandes superordenadores. Sin embargo, no se ha demostrado que el proyecto se haya beneficiado científicamente por el orden de magnitud en la cantidad de equipos que se utilizan, muchos fuera de casa (la intención original era usar de 50,000 a 100,000 ordenadores domésticos).

El software de computación distribuida SETI@home se ejecutaba tanto como protector de pantalla o de manera continua, mientras que un usuario trabaja, haciendo uso de la capacidad del procesador que de otra manera no se empleen. La plataforma de software inicial, que ahora se conoce como "Classic SETI@home", funcionó entre el 17 de mayo de 1999 hasta el 15 diciembre de 2005. Este programa sólo era capaz de ejecutar SETI@home; y fue reemplazado por BOINC, que también permite a los usuarios contribuir en otros proyectos de computación distribuida y al mismo tiempo ejecutar SETI@home. La plataforma BOINC permitirá también hacer pruebas de otros tipos de señales. La supresión de la plataforma de SETI@home Classic hizo que los equipos Macintosh con versiones previas al sistema Mac OS X no fueran aptos para participar en el proyecto. El 03 de mayo 2006 comenzó la distribución de unidades de trabajo para una nueva versión de SETI@home llamada "SETI@home mejorada". Dado que los ordenadores tienen ahora el poder de cómputo para un trabajo más intenso

que cuando el proyecto comenzó, esta nueva versión es dos veces más sensible con respecto a las señales de Gauss y de algunos tipos de señales en impulsos que el software SETI@home (BOINC) original. Esta nueva aplicación ha sido optimizada para el punto en que se ejecutará más rápido en algunas unidades de trabajo que las versiones anteriores. Además, algunas distribuciones de las aplicaciones de SETI@home han sido optimizados para un determinado tipo de CPU. Estos se conocen como "ejecutables optimizados" y se ha encontrado que corren más rápido en sistemas con esa CPU específica. A partir de 2007, la mayoría de estas aplicaciones están optimizadas para los procesadores Intel (y sus correspondientes conjuntos de interrupciones). Los resultados del procesamiento de datos normalmente se transmiten automáticamente cuando el equipo está conectado a Internet, aunque también puede ser programado para conectarse a Internet cuando sea necesario.

Entre los principales patrocinadores del proyecto se encuentran:

- The Planetary Society
- Sun Microsystems
- Intel
- ProCurve Networking
- SnapAppliance
- Quantum
- XILINX

c) EINSTEIN@HOME [2]

Al igual que SETI@Home, EINSTEIN@Home es un proyecto de computación voluntaria que analiza ondas gravitacionales producidas por fuentes de ondas continuas, las cuales pueden incluir pulsares. El nombre proviene del científico alemán, Albert Einstein, quien a principios del siglo XX predijo la ocurrencia de estas ondas gravitacionales.

El proyecto fue lanzado oficialmente el 19 de Febrero del 2005 como contribución por parte de la Sociedad Americana de Física para el World Year of Physics 2005.
El objetivo científico del proyecto es la búsqueda de fuentes de radiación gravitacional de onda continua. El éxito en la detección de ondas gravitacionales constituiría un hito importante en la física, ya que nunca antes se ha detectado un objeto astronómico únicamente por radiación gravitacional.
La información es obtenida mediante dos fuentes, por el Laser Interferometer Gravitational-Wave Observatory, y mediante GEO 600, otro detector de ondas gravitacionales.

d) ROSETTA@HOME [2]

Rosetta@Home es un proyecto orientado a determinar las formas tridimensionales de las proteínas a través de investigaciones científicas experimentales que a la larga podrían llevar a descubrir curas para las más grandes enfermedades humanas, como el VIH, la malaria y el cáncer.
Este y todos los proyectos antes mencionados son parte de la computación voluntaria, donde usuarios alrededor del mundo donan su tiempo de computo desperdiciado a diversos proyectos, los cuales pueden ser elegidos, por ejemplo, por interés en el tipo de proyecto planteado.

e) Otros Proyectos

Muchos otros proyectos, en distintas áreas de investigación, utilizan la tecnología BOINC, donde cabe destacar los siguientes:

- Matemáticas, Juegos de Estrategia, 3D, Informática
- SZTAKI DesktopGrid
- Chess960@Home
- Rectilinear Crossing Number
- Riesel Sieve

- VTU@Home
- Render Farm
- Prime Grid
- Xtrenelab
- ABC@Home
- DepSpi
- Biología y Medicina
- Malariacontrol.net
- Predictor@Home
- World Community Grid
- SIMAP
- Tanpaku
- Ralph@Home
- Docking@Home
- Project Neuron
- Proteins@Home
- Astronomía, Física y Química
- QMC@Home
- LHC@Home
- Spinhenge@Home
- Leiden Classical
- Hash Clash
- Ufluids
- Orbit@Home
- Ciencias de la Tierra
- Climateprediction
- BBC Climate Change Experiment
- Seasonal Attribution Project

2.1. Terracota [2]

Terracota es una infraestructura de software open source que permite ejecutar una aplicación hecha en Java en cuantos computadores sean necesarios para tal fin, sin tener que elaborar un código especifico para poder ser ejecutado en un clúster convencional.

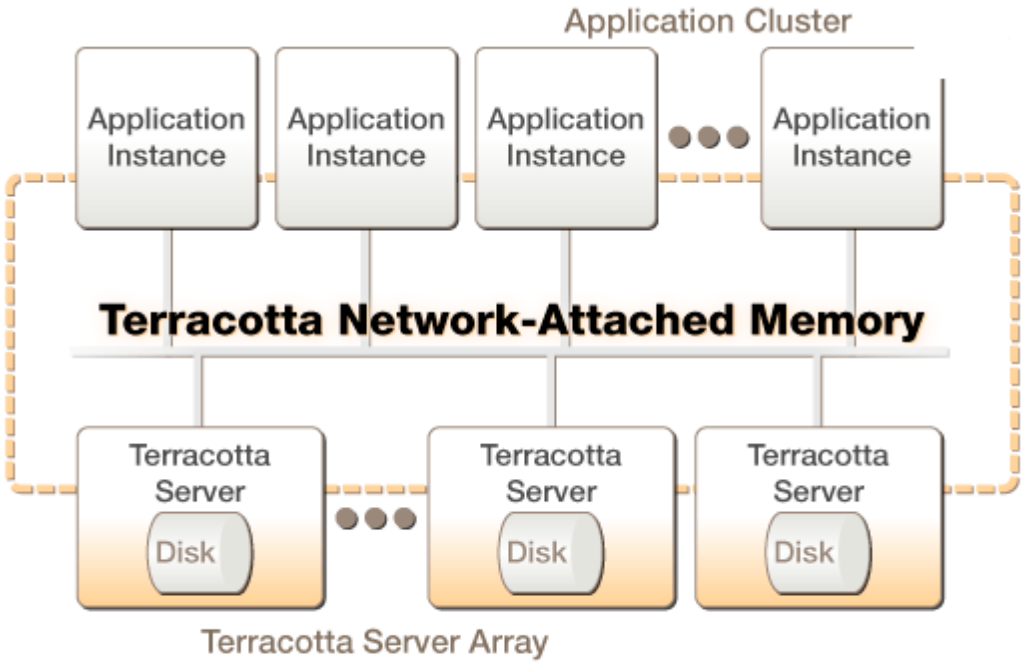
La arquitectura de Terracota tiene dos elementos principales, los nodos cliente y el Terracota Server Array.

Cada nodo cliente corresponde a un proceso Java dentro del clúster. Estos nodos ejecutan una Maquina Virtual Java estándar, logrando cargar a Terracota mediante sus librerías; todo esto ocurre cuando la Maquina Virtual Java se inicializa.

El Terracota Server Array provee una inteligencia de clustering tolerante a fallos, además de instalaciones de alto rendimiento. Cada instancia del Servidor Terracota dentro del array es 100% procesos Java.

Terracota utiliza la tecnología Network-Attached Memory (NAM) para permitir realizar el clustering con Maquinas Virtuales Java. Como se muestra en la imagen siguiente, Terracota es la Maquina Virtual Java para la aplicación a ejecutar dentro del clúster, mientras que para la verdadera Máquina Virtual Java, Terracota es la aplicación.

Figura 2: Arquitectura Terracota



Fuente: ADMINISTRADOR DE PROYECTOS DE GRID COMPUTING QUE HACEN USO DE LA CAPACIDAD DE CÓMPUTO OCIOSA DE LABORATORIOS INFORMÁTICOS, Martín Alberto Ibérico Hidalgo, pág. 19.

Con Terracota, un cambio en una Maquina Virtual Java se ve reflejado casi instantáneamente en las demás maquinas virtuales que necesitan saber del cambio producido.

3. ESTILOS ARQUITECTURALES DE LOS SISTEMAS DISTRIBUIDOS

Los sistemas distribuidos son comúnmente piezas complejas de software, cuyos componentes, por definición se encuentran dispersos en múltiples maquinas, la forma de coordinar y controlar esta complejidad es crucial para la correcta organización del sistema.

La organización de un sistema distribuido se concentra mayormente en los componentes de software que lo constituyen. Las arquitecturas de software describen como estos componentes están organizados y cómo interactúan, y los estilos arquitecturales definen la forma como estos componentes se conectan, como intercambian información y como son configurados en conjunto con el sistema.

A continuación se describe una línea de tiempo de los diferentes estilos arquitecturales, ahondando en el contexto en que se originaron, es decir las problemáticas que abordaron, la forma como se dio respuesta a esta y las tecnologías que los sustentaron.

3.1. Ad Hoc Network Programming

A mediados de los años 80's**,** el surgimiento de los microprocesadores y las redes de datos, abrió la posibilidad de comunicar a las computadoras y sus componentes de software, entre sí, lo cual llevo a la creación de los mecanismos de comunicación de interprocesos (IPC), que permiten la cooperación de procesos con diferentes espacios de memoria aún cuando estos se encuentran en diferentes maquinas.

Los principales mecanismos IPC son:

- **Sockets**: Es un concepto abstracto que hace referencia a un punto de comunicación bidireccional, a través del cual dos programas pueden intercambiar flujos de datos de manera confiable y ordenada.

- **Memoria Compartida:** Los procesos que se ejecutan en una misma máquina tienen espacios de direcciones diferentes, por lo tanto, el espacio de memoria utilizado por un proceso no podrá ser accedido por ningún otro. La memoria compartida permite la cooperación entre procesos, al crear un espacio en memoria en el cual se pueda compartir información. Esta área puede ser creada por un proceso y posteriormente escrita y leída por cualquier número de procesos.

- **Tuberías (pipes)**: consiste en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo.

Los sistemas diseñados con base en este estilo arquitectural, actualmente son escasos debido al gran esfuerzo requerido para su implementación y mantenimiento, su poca flexibilidad y su gran dependencia del ambiente de operación.

3.2. Comunicación Estructurada

La comunicación estructurada nace como respuesta a la necesidad de desacoplar los aplicativos de los mecanismos IPC, creando mecanismos de comunicación de alto nivel a través del encapsulado de detalles de bajo nivel, tales como la lectura y escritura de bits, bytes, etc. De esta forma el estilo de comunicación estructurada se asemeja mucho a los estilos de programación de aplicaciones, implicando el uso de tipos y estructura de datos.

En la historia de los sistemas distribuidos, el ejemplo más significativo de este tipo de arquitectura son los RPC o Llamadas a Procedimientos Remotos, como es el

caso de Sun RPC, DCOM, y DCE. Estas plataformas permiten la colaboración en aplicaciones distribuidas a través de la invocación de funciones entre ellas, las cuales reciben parámetros y retornan el resultado a quien la invoca, lo cual permitió a los desarrolladores familiarizarse con este tipo de comunicación debido a sus semejanza con la programación orientada a objetos.

Las arquitecturas basadas en comunicación estructurada resolvieron una gran variedad de problemas inherentes a los sistemas distribuidos, y otras que no pudieron ser resueltas, las cuales se describen a continuación:

- **Transparencia en la ubicación de los componentes:** Dentro de un sistema distribuido, idealmente, la comunicación con recursos locales o remotos debería realizarse con el mismo modelo de programación, lograr esta independencia en la ubicación de los recursos requiere que el código sea separado de detalles específicos de las ubicaciones de la implementación.

- **Despliegue y redespliegue flexible de componentes:** En muchas situaciones el despliegue original de un componente puede dejar de ser la mejor manera de solucionar la problemática para la que fue creada, tales situaciones como el mejoramiento de hardware, la incorporación de nuevos nodos al sistema o la aparición de nuevos requerimientos, hacen necesario el redespliegue del componente, que en un sistema distribuido debe realizarse sin romper la continuidad del sistema ni mucho menos el apagado del mismo.
- **Integración con Legacy code:** Los sistemas distribuidos por diferentes razones, de costo y tiempos, deben ser capaces de integrar elementos y aplicaciones que no fueron desarrolladas para trabajar en un ambiente distribuido.

- **Componentes heterogéneos**: Una de las características de un sistema distribuido ideal es que sus componentes puedan ser desarrolladas en

diferentes lenguajes y ambientes, permitiendo ser integrados en un solo y coherente sistema distribuido.

3.3. Middleware

Debido a la gran variedad de dispositivos de computo y software que existen, la comunicación y cooperación entre estos, se ha convertido en una necesidad que día a día toma mayor relevancia. Como respuesta a esta necesidad aparece el concepto de middleware.

Un middleware desarrolla una infraestructura distribuida de software dedicado, el cual reside entre las aplicaciones, y el sistema operativo, las bases de datos, la red, etc., que permite construir sistemas distribuidos con características que con arquitecturas tradicionales no sería posible.

3.3.1. Middleware de Componentes

Un componente de software puede definirse como una pieza no trivial de software, un módulo, un paquete o un subsistema que completa una función clara, tiene un límite claro y puede ser integrado en una arquitectura bien definida.

La aplicación de este concepto en sistemas distribuidos es lo que permite el nacimiento de este estilo arquitectural, el cual es una extensión de los middleware de objetos distribuidos y es una respuesta a las limitaciones que contienen estos, aborda las limitaciones funcionales permitiendo a objetos agruparse de manera coherente en componentes, a través del uso de múltiples interfaces, y la definición de mecanismos estandarizados necesarios para la ejecución de dichos componentes servidores de aplicaciones genéricas. Las limitaciones en el despliegue de software y las configuraciones son superadas con la especificación de infraestructuras para encapsulamiento, personalización, ensamble y distribución de componentes sobre sistemas distribuidos.

Algunos ejemplos de esta tecnologías son los Enterprise JavaBeans y CORBA Component Model.

La aparición de este tipo de middleware permitió la automatización de ciertos aspectos en el ciclo de vida de las aplicaciones de software, tales como: el empaquetamiento, el ensamblaje y el despliegue, lo que ha facilitado construcción de aplicaciones, de forma más rápida y robusta que el middleware de objetos distribuidos.

3.3.2. Middleware orientados al mensaje

Las arquitecturas que se han estudiado en este capítulo, poseen un modelo común de comunicación de tipo cliente - servidor, en el cual el cliente realiza peticiones que son respondidas por el servidor, lo que implica que la comunicación solo puede ser iniciada por el cliente, por lo cual este modelo, resulta insuficiente para cierto tipo de aplicaciones distribuidas, que deben reaccionar a estímulos externos o eventos, y en los cuales el trabajo colaborativo entre maquinas es importante, como es el caso de los sistemas de control ó sistemas de negociación de bolsa de valores.

Los middleware orientados al mensaje, nacen como una solución a este tipo de sistemas, ofreciendo una pasarela de mensajes, que permite a las aplicaciones intercambiar información asíncronamente y de forma desacoplada, evitando el bloqueo durante la espera de la respuesta. Además proveen frecuentemente propiedades transaccionales como colas de confiabilidad y persistencia hasta que el consumidor pueda recoger los mensajes.

Hay dos subtipos de middleware orientados a mernsajes, los de mensajería punto a punto y los de publicación/suscripción. En los primeros un sistema de mensajes redirige los mensajes desde el emisor hasta la cola de mensajes del receptor, donde el middleware proporciona un depósito de mensajes, permitiendo el desacoplamiento del envío y la recepción, también proporcionan abstracciones

que facilitan la extracción de los mensajes y el manejo asíncrono y paralelo de las operaciones necesarias para el procesamiento de los mismos.

Para el caso de los middleware de publicación/suscripción, están basados en la comunicación anónima, donde hay un débil acoplamiento entre el publicador y el consumidor, y por lo tanto no conocen la existencia del otro, lo cual permite que múltiples subscriptores reciban datos de un publicador. Este tipo de middleware permite a las aplicaciones correr en nodos distintos y realizar eventos de escritura-lectura sobre un espacio global de datos en un sistema distribuido. Las aplicaciones usan este espacio global para compartir información con otras aplicaciones, declarando su intención de producir eventos, lo cual frecuentemente es categorizado en tópicos de interés, así si otra aplicación que dese consumir debe declarar su intención en el espacio global.

Los elementos del middleware de publicación/suscripción son separados en los siguientes roles:

- Publicadores: son las fuentes de eventos, producen eventos en tópicos específicos que son propagados en el sistema.
- Suscriptores: son quienes reciben los eventos, es decir reciben información de tópicos que les interesan.
- Canales de eventos: Son los componentes del sistema que propagan los eventos del publicador al suscriptor.

3.4. Arquitectura Orientada a Servicios y Servicios Web

Una Arquitectura orientada a servicios (SOA) es un estilo de organización y utilización de las capacidades distribuidas que pueden ser controlados por las diferentes organizaciones o propietarios. Por lo tanto, proporciona un modo uniforme para ofrecer, descubrir, interactuar y utilizar estas capacidades débilmente acopladas y servicios de software interoperables para soportar las necesidades de los procesos de negocios y los usuarios de la aplicación.

En una arquitectura orientada a servicios, se diferencia claramente tres elementos básicos:

- **Servicio:** conjunto de actividades que son ofrecidas por una o más entidades, y que responden a necesidades especificas de un grupo de clientes, los cuales son los encargados de consumirlo.
- **Proveedor:** Ente encargado de implementar, poner en marcha y presentar el servicio.
- **Consumidor:** Ente que consume los servicios.

SOA, propone un conjunto de directivas para la elaboración de software en ambientes distribuidos, pero como tal no específica que tecnologías y estándares se usaran en la implementación específica de un sistema, es en este punto que los servicios web han tomado importancia, como la principal variante de SOA.

Los servicios web son una tecnología que permite encapsular lógica de aplicación en servicios, y que a través de estándares dictados por World Wide Web Consortium (W3C), que define los principales aspectos de dichos servicios, entre los mas importante tenemos el lenguaje de descripción de servicios web o WSDL.[3]

4. GRID COMPUTING

4.1. Historia

El primer uso de una grilla se dio debido a la necesidad de HEP (High-Energy Physics) de diseños de modelos computacionales para experimentos con LHC (Large Hadron Collider – Gran Colisionador de Hadrones) que sería la siguiente generación de acelerador de de partículas de la CERN en Ginebra.

La Organización Europea para la Investigación Nuclear (CERN, sigla que corresponde a su nombre en francés: Conseil Europeen poor la Recherche Nucleaire) se encuentra en Suiza, cerca de Ginebra y es el mayor laboratorio de investigación en física de partículas a nivel mundial; entre sus principales desarrollos se encuentran la invención de WordWideWeb en 1990 y el sistema de almacenamiento masivo LHC.

El Gran Colisionador de Hadrones es un acelerador de partículas que fue creado para colisionar haces de hadrones (partícula subatómica que experimenta interacciones nucleares fuertes, por ejemplo los neutrones y protones) con el propósito de examinar la validez y limites del Modelo Estándar, teoría que describe las relaciones entre las interacciones fundamentales conocidas entre partículas elementales que componen toda la material; dicho modelo es el marco teórico de la física de partículas.[4]

Básicamente es un anillo de imanes superconductores de 27 km de circunferencia que acelera opuestos haces de protones o iones de plomo y les choca de frente a elevadas energías en el centro de los cuatro grandes detectores. El colisionador y sus experimentos se encuentran a 100 metros de profundidad, dos de ellos, ATLAS (A Toroidal LHC Apparatus) y CMS (Compact Moun Solenoid) son enormes detectores de uso general que tienen como objetivos descubrir las dimensiones de la materia oscura, mientras que otros como ALICE (A Large Ion Collider Experiment – Experimento del Gran Colisionador de Iones) estudian las colisiones de iones de plomo para entender el plasma de quark-gluon, un estado

de la materia que se cree que ha existido desde después de la explosión del BigBang.

Debido a las necesidades creadas en el proyecto LHC, los científicos de CERN iniciaron el proyecto LCG (LHC Computing Grid) para crear un ambiente que permita el gran almacenamiento y el poder computacional que requieren los experimentos con LHC. Se empezó con el uso de datagrids para entender su tecnología y la ayuda que podría brindar para resolver los problemas de la HEP. Se estimaba que dichos experimentos producían aproximadamente 15 Petabytes (15 millones de GB) de información por año que serian almacenados y distribuidos a miles de científicos en cientos de laboratorios alrededor del mundo, mientras que 100.000 procesadores se necesitarían en los primeros años para procesar y analizar la información.

La primera meta de LCG era integrar la tecnología de las grillas que habían sido probadas en ambientes de prueba con los ambientes computaciones existentes de los centros de HEP. En dos años LCG construyo cerca de 40 sitios que permitían incorporarse con los experimentos en trabajos de simulación mientras permitían aprender a construir las mejores aplicaciones utilizando los servicios de que proporcionaban las grillas.

LCG evoluciono en WLCG (WorldWide LCG), una unión que ahora incluye 36 países. Los centros computacionales que hacen parte de WLCG son los que definen la cantidad de recursos y la cantidad de servicios a utilizar. Una indicador principal para WLCG es la habilidad de enviar información de CERN a 11 centros computacionales regionales que proporcionan servicios de análisis de dicha información y a su vez envían los datos procesados a los centros más pequeños donde los físicos los analizan. Este proceso alcanza cantidades de información de hasta 200 terabytes por día.

Desde el 2004, WLCG ha estado corriendo una serie de pruebas para preparar el inicio del LHC, logrando mejoras en la funcionalidad, confiabilidad y nivel de servicios.

Existen dos diferentes proyectos que son los pilares de los centros computacionales de WLCG: EGEE (Enabling Grids for E-sciencE) en Europa y OSG (OpenSourceGrid) en los Estados Unidos.

EGEE actualmente sirve a una amplia gama de comunidades científicas europeas y es la más grande y más utilizada infraestructura de grilla a nivel mundial. En sus experimentos, EGEEE le ha probado a E-science la viabilidad de la estructura de una grilla computacional con escalabilidad, manejo de datos y computación distribuida para satisfacer sus requerimientos. Hoy por hoy, el servicio de producción de la EGEE tiene acceso a más de 20 petabytes de almacenamiento y 80.000 unidades de procesamiento, los cuales se espera que aumenten dramáticamente durante el año debido a los recursos extras de los experimentos de LHC y a las nuevas aplicaciones online. Para compartir recursos y acceder a base de datos comunes a través de la grilla sin quebrantar su modelo de seguridad EGEE creó el concepto de Organizaciones Virtuales.

Una organización virtual es un conjunto dinámico de individuos y/o instituciones definidas en torno a un conjunto de normas de reparto de recursos y condiciones; todas estas organizaciones comparten algo en común entre ellos, incluidos los intereses y necesidades, pero pueden variar en tamaño, alcance, duración y estructura, aunque el concepto ha sido definido de varias formas a través de los años; Gerardine DeSanctis y Peter Monge, teóricos organizacionales las definieron como: “una colección de diversas entidades geográfica, funcional o culturalmente distribuidas que están conectadas por formas electrónicas de comunicación y dependen de dinámicas relaciones para su coordinación”. Ian Foster, Carl Kesselman y Steven Tuecke las definieron como “flexibles, seguros y coordinadas formas de compartir información a través de colecciones dinámicas de individuos, instituciones y recursos”.

Europa ha invertido mucho en E-science a nivel tanto regional como nacional con grandes resultados. Con el apoyo de la comisión europea se ha convertido en un líder mundial en el campo de la tecnología Grid pero aunque la EGEE ha sido un

éxito no es el único modelo, futuros desarrollos continuaran aumentando la complejidad y generando nuevos retos. [8]

4.2. Evolución

La computación Grid ha evolucionado a partir de la computación de alto rendimiento y se distribuye en la década de 1990. La evolución se debió principalmente a la necesidad siempre creciente de los recursos informáticos, la disponibilidad de tecnología cada vez más potentes para la creación de redes, servidores, middleware y aplicaciones, y el desarrollo y la aceptación generalizada de la Internet y la World Wide Web. Hoy, como antes con la Web, la computación distribuida se puede ver cómo crece en tres etapas: la primera es la evolución de la conceptualización misma, el crecimiento de la idea en sí, expandiendo sus características y propiedades, se basa en investigaciones y nuevos desarrollos; la segunda etapa es de interfaces y aplicaciones grid, la creación de middlewares y aplicaciones que permitan la conexión entre los recursos de la grilla permitiendo su correcto funcionamiento; y la ultima, es la etapa de comunicación global, la cual está sucediendo debido a que grandes empresas de tecnología (Sun, IBM, Intel, etc) han empezado proyectos bajo el uso de computación grid enfocados primordialmente en reducir costo y complejidad para cada una de las mismas.

4.2.1. Primera Etapa

Aunque difícil de creer, la idea de computación Grid ha sido concebida desde los primeros días de la informática en sí, ya en 1969 se encuentran referencias de lo que sería la visión de una infraestructura Grid:

“Probablemente vamos a ver la extensión de los servicios informáticos, que, como los actuales servicios de electricidad y teléfono, darán servicios a los hogares y oficinas en todo el país.”

Como hoy por hoy se puede apreciar, esta visión ha sido implementada gradualmente desde entonces.
Influenciado por la investigación en computación paralela y distribuida en la década de 1990 el término “Grid” poco a poco se popularizo. El termino como tal fue formalizado por primera vez en 1999 en el libro: “The Grid: Blueprint for a New Computing Infraestructure” (“Grid: Proyecto para una nueva infraestructura computacional.”)

La definición dada fue:”Un Grid computacional es una estructura de software y hardware que proporciona fiabilidad, acceso constante, generalizada y de alta capacidad computacional.” [5]
Esta definición se centra en una infraestructura de software y hardware; a principios de ese mismo año sin embargo, Von Laszewski había proyectado que la comunidad estaba en la necesidad de una infraestructura extendida como una “infraestructura informática compartida de software, hardware y recursos de conocimiento”.
Dichos recursos de conocimiento abarca no solo las bases de datos sino también los seres humanos. Las implicaciones de esta distinción son importantes, de hecho, teniendo en cuenta esto en el año 2000, Von Laszewski define el término “Enfoque Grid” en el que señala que la computación grid promueve una perspectiva de la realización de colaboraciones que no se detiene en el software y hardware, sino que permita una visión de una infraestructura compartida entre los seres humanos, que entre sus principales ventajas tiene la colaboración mundial en el ámbito de los negocios. [5]

El progreso de la computación Grid ha permitido lógicamente la evolución de las características de la misma junto a los beneficios que brinda, aunque los artículos recientes se han centrado principalmente en resaltar el beneficio de un mejor aprovechamiento de recursos infrautilizados de computación (algo que de por si proporciona un gran ahorro) no es el único que una Grid ofrece, existen otros tales como:

- Fácil Acceso: Transparente, acceso remoto, seguro.
- Vitalización de recursos: Acceso a la computación y servicios de datos no a los propios servidores sin preocuparse por la infraestructura.
- Compartir recursos: Permite la colaboración de equipos virtuales a través de internet para trabajar conjuntamente en una tarea compleja.
- Conmutación por error: En caso de fallo del sistema, las aplicaciones pueden migrar y se reinicia/continúa de forma automática.
- Heterogeneidad: En redes grandes y complejas, los recursos son heterogéneos (plataformas, sistemas operativos, dispositivos, software, etc.), los usuarios pueden elegir el sistema más adecuado para su aplicación especifica o el software de la grid transparentemente elige el más adecuado y el que menor recursos necesite.

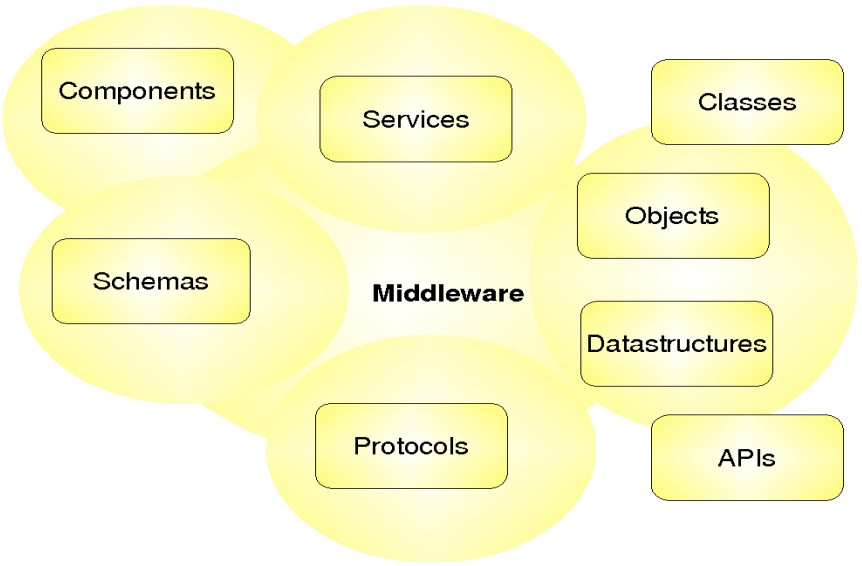
4.2.2. Segunda Etapa

Por otro lado, la evolución de programas para computación Grid, naturalmente refleja la evolución del concepto Grid como tal. Estos programas deben lidiar con los diferentes desafíos de gestión que Grid ofrece, los cuales se basan principalmente en cuatro puntos específicos: (1) exponer la grilla a los usuarios de tal forma que sea totalmente transparente para ellos, donde de hecho, los usuarios puede que no sean conscientes de estar usándola; (2) desarrollar software que permita a los desarrolladores de aplicaciones grid integrarlas fácilmente; (3) desarrollar software para permitir la creación de sofisticadas infraestructuras middleware para ser reusadas por el desarrollador de la aplicación, y (4) desarrollar aplicaciones que cómodamente se comuniquen con otros software y el hardware de la grilla. [5]

En cuanto a la tendencia actual en el desarrollo de middleware, uno podría tener la impresión de que todo el software de computación Grid desarrolla servicios Grid, sin embargo, esto no es cierto. Si bien el desarrollo de estos servicios es parte del

middleware, los servicios todavía deben ser implementados a través de las prácticas comunes de ingeniería de software para exponer la lógica. Por lo tanto, además de los servicios Grid es necesario para producir los esquemas, herramientas, APIs, modelos de objetos y marcos de programación que hacen que la solución de aplicaciones de extremo a extremo de una realidad (ver Figura 3).

Figura 3: Middleware para soportar las necesidades de una variedad de tecnologías con el fin de dar cabida a una amplia variedad de usos.



Fuente: The Grid Idea and its Evolution; Gregor Von Laszewski, Laboratorio nacional de Argonne, USA. Pag 5.

La definición exacta del Middleware especifica que es la capa de software situado entre el sistema operativo y las aplicaciones, proporcionando una variedad de servicios requeridos por una aplicación para que funcione correctamente. Recientemente, el middleware se ha re-emergido como un medio de integración de aplicaciones de software que se ejecuta en entornos distribuidos heterogéneos. En una grilla, el middleware se utiliza para ocultar la naturaleza heterogénea y proporcionar a los usuarios y aplicaciones con un entorno homogéneo y sin

fisuras, proporcionando un conjunto de interfaces estandarizadas para una variedad de servicios.

Hay un número creciente de proyectos relacionados con la grilla, se ocupan de áreas tales como infraestructura, servicios básicos, la colaboración, aplicaciones específicas y los portales de dominio. Estos son algunos de los más importantes hasta la fecha:

a) Globus

Proporciona una infraestructura de software que permite a las aplicaciones distribuidas manejar los recursos informáticos heterogéneos como una sola máquina virtual. El proyecto Globus es un multi-institucional esfuerzo americano de investigación que tiene por objeto permitir la construcción de grillas computacionales. Una grilla computacional, en este contexto, es una infraestructura de hardware y software que proporciona acceso fiable, constante y generalizado de alta capacidad de cálculo, a pesar de la distribución geográfica de los recursos y los usuarios. Un elemento central del sistema de Globus es el Globus Toolkit, que define los servicios básicos y las capacidades necesarias para construir una red computacional. El kit de herramientas se compone de un conjunto de componentes que implementan los servicios básicos, tales como la seguridad, localización de recursos, gestión de recursos, y las comunicaciones.
Es necesario para las grillas computacionales soportar una amplia gama de aplicaciones y paradigmas de programación. En consecuencia, en lugar de proporcionar un modelo de programación uniforme, como el modelo orientado a objetos, el Globus Toolkit ofrece una bolsa de servicios que los desarrolladores de herramientas o aplicaciones pueden utilizar para satisfacer sus propias necesidades. Esta metodología sólo es posible cuando los servicios son distintos y tienen interfaces bien definidas (API) que pueden ser incorporados en aplicaciones o herramientas en forma incremental.

Globus está construido como una arquitectura de capas en las que servicios de alto nivel mundial se basan en los servicios esenciales de base de bajo nivel local. El Globus Toolkit es modular, y una aplicación puede aprovechar las características Globus, como la gestión de recursos o la infraestructura de información, sin usar las bibliotecas de comunicación Globus. El Globus Toolkit en la actualidad está conformado por:

- GRAM - Globus Toolkit Resource Allocation Manager: (Globus Toolkit Gerente de Asignación de Recursos) Un protocolo HTTP utilizado para la asignación, vigilancia y control de los recursos computacionales.
- Una versión extendida del protocolo de transferencia de archivos, GridFTP, que se utiliza para acceder a los datos, las extensiones incluyen el uso de protocolos de conectividad de capa de seguridad, acceso a los archivos parciales, y la gestión de paralelismo para transferencias de alta velocidad.
- GSI - Grid Security Infraestructure :
  (Infraestructura de Seguridad de la Grilla). Autenticación y servicios relacionados con la seguridad
- GASS – Globlal Access to Secondary Storage :
  (Acceso Global a un Almacenamiento Secundario) Acceso remoto a los datos a través de interfaces secuenciales y paralelos, incluyendo una interfaz para GridFTP.
- GEM - Globus Executable Management:
  (Gestión de Ejecutables Globus) Construcción, almacenamiento en caché y ubicación de los ejecutables.

b) Legion

Legion es un objeto basado en un meta-sistema, fue desarrollado en la Universidad de Virginia. [6] Legion proporciono la infraestructura de software para que un sistema heterogéneo, distribuido geográficamente, con máquinas de alto

rendimiento pueda interactuar a la perfección. Legion intento proporcionar a los usuarios, en sus estaciones de trabajo, con una única infraestructura integrada independientemente de su tamaño, la ubicación física del lenguaje, y el sistema operativo subyacente.

Legion difería de Globus en su enfoque para la provisión de un entorno Grid: se encapsula todos sus componentes como objetos. Esta metodología tiene todas las ventajas normales de un enfoque orientado a objetos, tales como la abstracción de datos, la encapsulación, herencia y polimorfismo.

c) Jini

Jini está diseñado para proporcionar una infraestructura de software que se puede formar en un entorno de computación distribuido que ofrece la red plug and play. Una colección de Jini procesos habilitados constituye una comunidad Jini - una colección de clientes y servicios de comunicación por todos los protocolos de Jini. En Jini, las aplicaciones normalmente se escriben en Java y se comunican usando la invocación a métodos remotos de Java (RMI). A pesar de que Jini está escrito en Java puro, ni los clientes ni los servicios de Jini se ven obligados a ser puro Java. Pueden incluir envoltorios de Java alrededor de código no-Java, o incluso estar escritos en otro idioma por completo. Esto permite a una comunidad Jini extenderse más allá de los marcos normales de Java, los servicios de enlace y los clientes de una variedad de fuentes. [6]

d) Condor

Condor es un paquete de software para la ejecución de trabajos por lotes en una variedad de plataformas UNIX, en particular las que de otra manera sería inútil. Las principales características del Plan Condor son la ubicación automática de recursos, asignación de trabajo y la migración de los procesos. Estas características se aplican sin modificación de la base del kernel UNIX. Sin

embargo, es necesario que el usuario vincule su código fuente con las bibliotecas Condor. Condor supervisa la actividad de todos los recursos de computación que participan, las máquinas que están programadas para estar disponibles se colocan en un fondo de recursos el cual las asignan para la ejecución de los trabajos. [6]

e) Nimrod-G

Nimrod-G es una aplicación Grid que lleva a cabo la gestión de recursos y la programación de los parámetros de barrido y aplicaciones de tareas agrícolas. Consta de cuatro componentes: Un motor de las tareas agrícolas, un planificador, un despachador y agentes de recursos. [6]

El distribuidor utiliza Globus para el despliegue de agentes de Nimrod-G en los recursos remotos con el fin de gestionar la ejecución de los trabajos asignados. El planificador de Nimrod-G tiene la capacidad para liberar los recursos de la grilla y sus servicios en función de su capacidad, costo y disponibilidad. El planificador apoya la búsqueda de recursos, selección, programación y la ejecución de trabajos de usuario a los recursos remotos. Los usuarios pueden establecer el plazo dentro del cual sus resultados son necesarios y el corredor de Nimrod-G trata de encontrar los mejores recursos disponibles en la red y utilizarlos para cumplir con el plazo del usuario y tratar de minimizar los costos de la ejecución de la tarea.

f) Portales Grid

Un portal web permite a los científicos e investigadores de la aplicación acceder a recursos específicos de un dominio específico de interés a través de una interfaz Web. A diferencia de las portales comunes, un portal Grid también puede proporcionar acceso a los recursos de la red. Por ejemplo, en un portal Grid se puede autenticar a los usuarios, les permiten acceder a recursos remotos, ayudarles a tomar decisiones acerca de los trabajos de programación, y permite

acceder y manipular la información de los recursos obtenidos y almacenados en una base de datos remota. El acceso a un portal Grid también se puede personalizar mediante el uso de perfiles, que se crean y se almacenan para cada usuario del portal. Estos atributos y otros, hacen que los portales Grid sean los medios adecuados para aplicaciones de usuarios que necesitan acceso a los recursos de la grilla. Como por ejemplo el NPACI. [6]

NPACI es un portal de usuario que ha sido diseñado para ser un solo punto de acceso a recursos informáticos, para facilitar el acceso a los recursos que se distribuyen entre las organizaciones miembros y les permite ser visto ya sea como un sistema Grid o como máquinas individuales. [6]

Los dos principales servicios proporcionados por el NPACI son la información y acceso a los recursos y servicios de gestión. Los servicios de información están diseñados para aumentar la eficacia de los usuarios. Ofrece enlaces a:

- Documentación del usuario y la navegación
- Noticias de actualidad
- La formación y la consulta de información
- Los datos sobre las plataformas y aplicaciones de software
- Los recursos de información, tales como asignaciones de usuarios y cuentas

g) Sistemas Integrados

Con los nuevos componentes Grid surgieron una serie de grupos internacionales que comenzaron proyectos para integrarlos en sistemas coherentes. Estos proyectos se dedicaron crear un gran número de aplicación de alto rendimiento con enfoque en distintas áreas. Algunos de estos son Cactus 26, que es un ambiente de código abierto de solución de problemas para científicos e ingenieros, tiene una estructura modular que permite la ejecución de aplicaciones paralelas en una amplia gama de arquitecturas y desarrollo de códigos de colaboración entre

los grupos de distribución, se originó en la comunidad de investigación académica, en donde fue desarrollado y utilizado por una gran colaboración internacional de físicos y científicos de la computación para la simulación de un agujero negro; otro es DataGrid 27 un proyecto europeo dirigido por el CERN, está financiado por la Unión Europea con el objetivo de la creación de una grilla computacional y gran volumen de datos de recursos para el análisis de datos provenientes de la exploración científica, su aplicación es el Gran Colisionador de Hadrones (LHC), que operará en el CERN a partir de 2005 a 2015 y representa un salto adelante en materia de energía de rayos de partículas, la densidad y la frecuencia de colisión el cual es necesario para producir algunos ejemplos de las partículas que no habían sido descubiertos, como el bosón de Higgs. UNICORE 28 el cual incluye una uniforme y fácil de usar interfaz gráfica de usuario, una arquitectura abierta basada en el concepto de trabajo abstracto, una arquitectura de seguridad consistente, una mínima interferencia con los procedimientos administrativos locales, la explotación de las tecnologías existentes y emergentes a través del estándar de Java y las tecnologías Web, proporciona una interfaz para la preparación del trabajo y la sumisión seguro a los recursos superordenador distribuido. [6]

h) Computación Punto a Punto

Un método muy plausible para abordar los problemas de escalabilidad puede ser descrito como la descentralización, aunque esto no es una solución simple. El modelo tradicional cliente-servidor puede ser un cuello de botella en rendimiento y un punto de fallo único, pero todavía es prevalente porque la descentralización trae sus propios desafíos. En la computación P2P (Punto a Punto), las máquinas comparten datos y recursos, tales como ciclos de computación de repuesto y la capacidad de almacenamiento, a través de Internet o redes privadas. Las máquinas también pueden comunicarse directamente y administrar las tareas de computación sin usar servidores centrales. [6]

Una serie de sistemas de almacenamiento de P2P se están desarrollando dentro de la comunidad de investigación. En el contexto de grilla, estos plantean cuestiones importantes de seguridad y el anonimato.[6]

4.2.3. Tercera Etapa

La segunda etapa proporciona la interoperabilidad, que es esencial para lograr la computación a gran escala. Como nuevas soluciones Grid fueron exploradas otros aspectos de la ingeniería Grid se hizo evidente con el fin de crear nuevas aplicaciones grid capaces de reutilizar los componentes existentes y los recursos de información, y montar estos componentes de manera flexible. Las soluciones involucradas aumentan la adopción de un modelo orientado a servicios y cada vez más atención a los metadatos (dos características clave de los sistemas de tercera generación). De hecho, el enfoque orientado al servicio en sí tiene implicaciones en la información: el ensamblado flexible de los recursos en una aplicación Grid requiere información acerca de la funcionalidad, disponibilidad y las interfaces de los diferentes componentes, y esta información debe tener una correcta interpretación que debe ser procesada por la máquina.

Mientras que la grilla ha sido tradicionalmente descrita en términos de datos a gran escala y el cálculo, el cambio de enfoque en la tercera generación se desprende de las descripciones de nuevo. En particular, "la colaboración distribuida" y "organización virtual" fueron adoptadas en la "anatomía" de papel.
Hay un fuerte sentido de la automatización en los sistemas de esta etapa, cuando los seres humanos ya no pueden hacer frente a la escala y la heterogeneidad toca delegar en los procesos esta funcionalidad a través de secuencias de comandos, lo que conduce a la autonomía dentro de los sistemas. Esto implica una necesidad de coordinación, que a su vez, se debe especificar mediante programación en varios niveles, incluyendo descripciones de procesos. Del mismo modo, la mínima probabilidad de falla implica una necesidad de la recuperación automática, la configuración y la reparación no pueden ser tareas manuales. Estos requisitos se

asemejan a los de auto-organización y las propiedades curativas de los sistemas biológicos. Dichos requisitos constan de las siguientes características:

- Necesidades de conocimiento detallado de sus componentes.
- Se debe configurar y reconfigurar de forma dinámica
- Busca la optimización de su comportamiento para lograr su objetivo
- Es capaz de recuperarse de un mal funcionamiento
- Protegerse contra los ataques
- Ser consciente de su entorno
- Implementar estándares abiertos
- Hacer un uso optimizado de los recursos

Los sistemas grid actualmente en desarrollo están comenzando a exhibir muchas de estas características estandarizando así la comunicación global entre ellos:

a) Arquitectura Orientada a Servicios

En 2001, una serie de arquitecturas grid eran evidentes en una variedad de proyectos. Alrededor de este tiempo el modelo de servicios Web también fue ganando popularidad, prometiendo normas para apoyar un enfoque orientado a servicios. De hecho, una comunidad de investigación, la informática basada en agentes, ya había emprendido una amplia labor en este ámbito: los agentes de software pueden ser vistos como productores, consumidores y así como los agentes de los servicios. En total, se hizo evidente que el paradigma orientado a servicios siempre tuvo la flexibilidad necesaria para los aplicativos Grid de la tercera etapa. [6]

b) Servicios Web

La creación de estándares para los servicios web es una iniciativa liderada por la industria, con algunos de los estándares emergentes en diversos estados de

progreso a través de la World Wide Web Consortium (W3C). Las normas establecidas son:

- SOAP (Protocolo XML). SOAP proporciona un sobre que encapsula los datos XML para la transferencia a través de la infraestructura de la Web (por ejemplo, a través de HTTP, a través de cachés y proxies), con una convención de llamadas a procedimiento remoto (RPC) y un mecanismo de serialización de tipos de datos basados en XML Schema. SOAP es desarrollado por W3C, en colaboración con la Internet Engineering Task Force (IETF). [6]
- Servicios Web Description Language (WSDL). Describe un servicio en XML, utilizando un esquema XML, también existe una correlación con el framewrok de descripción de recursos (Resources Description Framework - RDF). En cierto modo, WSDL es similar a un lenguaje de definición de interfaces (IDL). [6]
- UDDI, una especificación para los registros de distribución de los servicios web, similar a los servicios de páginas amarilla. UDDI puede publicar, buscar y unir un proveedor de servicios describiendo y publicando los detalles del servicio en el directorio. También se basa en XML y SOAP. [6]

c) OGSA Framework

(Open Grid Service Architecture – Arquitectura abierta a servicios grid). Apoya la creación, mantenimiento y aplicación de los ensamblados de los servicios ofrecidos por las organizaciones virtuales (VO). Aquí un servicio se define como una entidad grid con capacidad de proporcionar una cierta capacidad, tales como los recursos computacionales, los recursos de almacenamiento, redes, programas y bases de datos. Se adapta el enfoque de los servicios web para cumplir con algunos requisitos específicos de la red. [6]

d) Agentes

El paradigma de la computación basada en agentes ofrece una perspectiva sobre los sistemas de software en el que las entidades suelen tener las siguientes propiedades:

- Autonomía, agentes operan sin intervención y tener cierto control sobre sus acciones y el estado interno.
- Sociales la capacidad, los agentes interactúan con otros agentes usando un lenguaje de comunicación de agentes.
- Reactividad, los agentes perciben y responden a su entorno.
- Proactividad, agentes dirigidos a un objetivo mostrar un comportamiento.

La informática basada en agentes se adapta particularmente bien a un entorno de cambio dinámico, donde la autonomía de los agentes permite el cálculo de adaptarse a las circunstancias cambiantes. Esta es una propiedad importante para la red de tercera generación. [6]

4.3. Fundamentos de la computación Grid

4.3.1. Definición

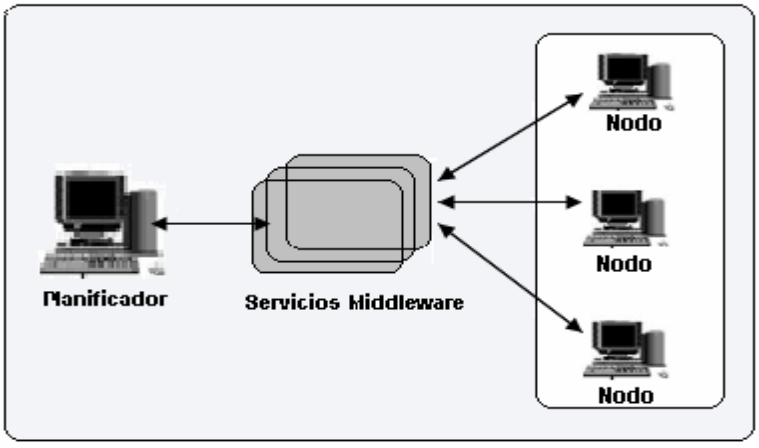
Grid es una infraestructura que permite el acceso y procesamiento concurrente de un programa entre varias entidades computacionales independientes. Un sistema paralelo y distribuido que permite la compartición, detección y agregación de recursos autónomos geográficamente distribuidos, de manera dinámica, en tiempo de ejecución, depende de su disponibilidad, capacidad, rendimiento, costo y requisitos de calidad del servicio de los usuarios.
Uno de los objetivos de un Grid consiste en integrar y optimizar, mediante middleware, el uso de recursos distribuidos de cálculo intensivo y de grandes bases de datos, como si estuvieran en un clúster local.

Las organizaciones que se interconectan por medio de un Grid mantienen sus propias políticas de seguridad y gestión de recursos. Esto significa que la tecnología usada para construir un Grid es complementaria a otras tecnologías, lo que permite aprovechar los recursos distribuidos en la Intranet de una organización, por ejemplo.

Las Grids integran el uso de diferentes tecnologías como redes, comunicación, computación e información que permiten proporcionar una plataforma virtual para la computación y gestión de los datos del mismo modo que Internet integra recursos para formar una plataforma virtual para la información. No importa si el usuario accede al Grid para utilizar un único recurso (ordenador, archivo de datos, etc.), o para utilizar varios recursos agregados como un ordenador virtual coordinado, en ambos casos, el Grid permite a los usuarios disponer, de manera uniforme, de una interfaz para los recursos, proporcionando una plataforma poderosa y comprensible para la computación global y la gestión de los datos.

La Grid se constituye como un pool de recursos computacionales geográficamente dispersos al que tenemos acceso. Una analogía utilizada frecuentemente se refiere a esta tecnología como la red eléctrica de recursos computacionales (ver Figura 4).

Figura 4: Funcionamiento básico de una grilla computacional.



Fuente: Computación Distribuida: Grid Computing, Benjamín Domínguez Hernández, Guatemala, Pág. 3.

El procesamiento de información o la ejecución de algún proceso en un nodo perteneciente a la Grid lo decide un planificador en función de las características de cada nodo y de cada partición. Cada nodo procesa la parte que le toca. Periódicamente podemos contactar con cada nodo, para asegurarnos de que todo va bien, e incluso realizar correcciones sobre la marcha. Cada nodo devuelve el resultado de la computación realizada.

4.3.2. Características generales y ventajas

Entre los beneficios de utilizar una arquitectura de Grid computing encontramos:

- Un ordenador central distribuye un proceso entre los ordenadores conectados a una red.

- El sistema aprovecha la capacidad de todos los ordenadores conectados. Cuando no están siendo utilizados al 100 por ciento por el usuario reciben tareas del ordenador central.

- Todos los recursos disponibles en la red son aprovechados, independientemente de su arquitectura y sistema operativo.

- Si el usuario está utilizando sólo una parte de la capacidad del ordenador, el resto se aprovecha para el cálculo.

- Si el usuario está utilizando todos los recursos o el ordenador se cae, la tarea se reasigna a otra máquina disponible.

- La aplicación es accesible desde cualquier ordenador de la red. Con un interfaz cómodo y sencillo se pueden añadir al sistema tantas máquinas como se desee.

- La conexión directa entre los PC s (P2P) evita la sobrecarga del ordenador central.

- La tecnología Grid es fácilmente escalable, creciendo según las necesidades de cada empresa.

- Transparente para el usuario que participa en la GRID.

- A diferencia de las redes convencionales que se centran en la comunicación entre dispositivos, la Grid computing aprovecha los ciclos de procesamiento no utilizado de todos los ordenadores conectados a una red de forma que se resuelven los problemas de las tareas que son demasiado intensivas para que las resuelva una única máquina.

- Integración de sistemas y dispositivos heterogéneos. Grid computing proporciona un conjunto de capacidades de integración horizontal que dirige de forma efectiva los recursos de toda una empresa, e incluso extienden la solución entre múltiples organizaciones. Por ejemplo, un científico que participe en una investigación Grid podría obtener el acceso a un único superordenador conectado a un laboratorio.

- Mejora el costo de los entornos operativos. A través de la visualización, compartición y gestión de recursos mediante funciones heterogéneas de tecnologías de la información, la Grid computing ayuda a simplificar los entornos operativos y su gestión reduciendo su administración y supervisión. Además, como consecuencia de fomentar la utilización eficiente de los recursos, Grid computing puede ayudar a las empresas a

construir una infraestructura de tecnologías de la información de costos efectivos que asegure la completa utilización de las inversiones en tecnología existente.

- Incrementa la capacidad de recursos para responder a las fluctuaciones de demanda. Permitiendo a las organizaciones de tecnologías de la información agregar recursos distribuidos y explotar una capacidad no utilizada, los Grids incrementan de forma importante la cantidad de recursos computacionales y de datos disponibles. La Grid computing ayuda a crear infraestructuras de tecnologías de la información que pueden responder rápidamente a oleadas inesperadas en el tráfico y uso de los recursos.

- Permite sacar ventaja de los recursos del Grid como una alternativa ante la recuperación de los desastres tradicionales, los departamentos de tecnologías de la información pueden mejorar la fiabilidad y disponibilidad de sus infraestructuras tecnológicas para aumentar la resistencia a una fracción del costo de los sistemas duplicados.

- Mejora el tiempo de obtención de resultados para nuevos productos y servicios. Aumentando la productividad y la colaboración, las organizaciones mejoran el tiempo en la obtención de resultados. Tanto si estos resultados incluyen llevar un nuevo producto al mercado más rápidamente, resolver un complejo problema de negocio más pronto, o realizar un análisis de datos en profundidad para lanzar un nuevo servicio. [10]

- Facilitan la colaboración y promueven la flexibilidad en las operaciones.

- La Grid puede agrupar no sólo recursos tecnológicos distintos, sino también a la gente. Facilitando que el personal pueda compartir, acceder y gestionar

la información, la tecnología Grid puede mejorar la colaboración y soporte a estrategias de globalización.

- Incrementan la productividad y permiten proporcionar a los usuarios finales un acceso sin restricciones a los recursos informáticos, de datos y de almacenamiento que necesitan, la tecnología Grid puede ayudar a las compañías a mejorar la gestión de recursos humanos.

- La utilización eficaz de los recursos existentes es una de las claves para minimizar costos. Los recursos y procesos organizacionales deben ayudar a asegurar la óptima utilización de los recursos informáticos, la tecnología Grid puede ayudar a las empresas a evitar las dificultades comunes de sobre-provisionamiento o incurrir en el exceso de costos para infraestructura.[10]

4.3.3. Organizaciones Virtuales

Como se menciono anteriormente las Organizaciones Virtuales (VO) son agrupaciones de recursos de varios individuos y/o organizaciones distintas que colaboran para alcanzar una meta común. Este fue un concepto creado por EGEE para compartir recursos y acceder a las bases de datos comunes a través de la grilla sin quebrantar el modelo de seguridad. [7]

El problema real y específico en que está justificado el concepto de Grid es coordinar la compartición de recursos y la resolución de un problema a través de organizaciones virtuales dinámicas multi-institucionales. El compartir al que nos referimos no es primordialmente el intercambio de archivos, sino sobre todo el acceso directo a las computadoras, al software, a los datos y a otros recursos, como es requerido por una gama de estrategias de resolución de problemas que requieren colaboración entre industria, ciencia e ingeniería.

Este compartir de recursos, debe ser altamente controlado, con los proveedores del recurso y los consumidores definiendo clara y cuidadosamente aquello que se comparte, lo que se permite compartir, y las condiciones bajo las cuales se puede compartir. El conjunto de instituciones y usuarios que comparten estos recursos y se someten a las reglas de compartición forman una organización virtual.

Los siguientes son ejemplos de VOs: proveedores de servicio a aplicaciones (ASPs, por sus siglas en ingles), proveedores de servicios de almacenaje, proveedores de ciclos de procesamiento, consultores comprometidos con un fabricante de autos para realizar la evaluación de un escenario durante la planeación para una fábrica nueva; miembros de un consorcio industrial que hace una oferta para un avión nuevo; un equipo de gerencia enfocado en paliar crisis, bases de datos y sistemas de simulación que se utilizan para planear la respuesta a una situación de emergencia.

Cada uno de estos ejemplos representa un enfoque para el procesamiento y la resolución de problemas basados en la colaboración e interacción de distintos recursos para cómputo y ambientes que necesitan grandes volúmenes de datos. Como también estos ejemplos demuestran, las VOs varían enormemente en su propósito, alcance, tamaño, duración, estructura, comunidad, y sociología. Sin embargo, el estudio cuidadoso de los requerimientos subyacentes de la tecnología nos conduce a identificar un conjunto de preocupaciones comunes y sus correspondientes requisitos. En detalle, vemos la necesidad de construir relaciones altamente flexibles para compartir recursos, evolucionando desde los sistemas cliente-servidor a arquitecturas peer-to-peer para los sofisticados y exactos niveles de control de cómo son utilizados los recursos compartidos, delegación, y uso de políticas locales y globales; para compartimiento de recursos variados, ejecución de programas, de archivos, y datos a computar, sensores, y redes; y para los diversos modos de uso, extendiéndose desde un solo usuario a múltiples y del funcionamiento sensible a coste-sensible y aplicaciones que por lo tanto están orientadas a calidad de servicio. Las tecnologías de computo

distribuidas actuales no tratan las preocupaciones y los requisitos anteriormente enumerados.

Las tecnologías de cómputo distribuidas tales como CORBA y Java permiten compartir recursos dentro de una sola organización. Los ambientes de cómputo distribuidos soportan el uso seguro de recursos compartidos a través de sitios, pero la mayoría de VOs lo encontraría demasiado pesado e inflexible. Los proveedores de servicios de almacenamiento (SSPs) y los proveedores de servicio a aplicaciones (ASP) permiten a las organizaciones cubrir sus requisitos de almacenaje y cómputo mediante outsourcing pero con ciertas restricciones: por ejemplo, los recursos de SSP son típicamente enlazados hacia un cliente por medio de una red privada virtual (VPN). El uso de compañías externas para realizar procesos de cómputo se apoya solamente sobre el acceso altamente centralizado a los recursos. En resumen, la tecnología actual no se acomoda a los diferentes usos de los recursos ni proporciona la flexibilidad y el control necesario para compartir los mismos y establecer VOs. [10]

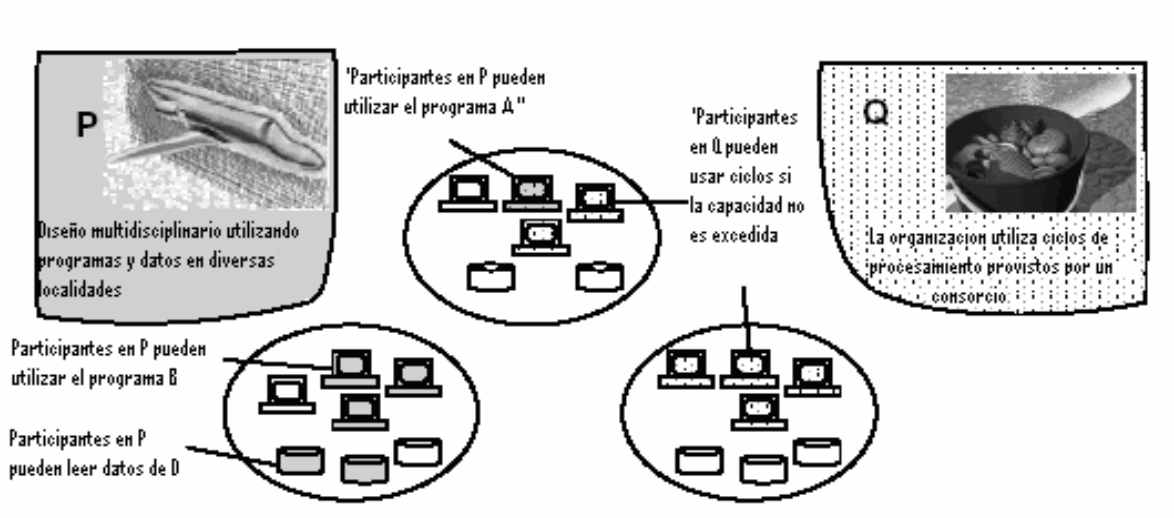
Es aquí donde las tecnologías GRID se incorporan al cuadro. Durante los últimos años, los esfuerzos de investigación y desarrollo de la comunidad GRID han producido protocolos, servicios, y herramientas que encaran los desafíos que se presentan cuando intentamos construir VOs escalables. Estas tecnologías incluyen soluciones de seguridad que soportan manejo de credenciales y políticas cuando los cómputos se realizan a través de múltiples instituciones; protocolos de gestión de recursos que proveen acceso seguro remoto a datos y recursos de computo y coasignación a múltiples recursos; protocolos de búsqueda de información y servicios que proveen configuración e información acerca del estado de los recursos, organizaciones, y servicios; y servicios de manejo de datos para localizar y transportar conjuntos de datos entre los sistemas del almacenamiento y las aplicaciones.

Debido a su enfoque dinámico, el uso de tecnologías Grid complementa las tecnologías distribuidas existentes. Por ejemplo, los sistemas de cálculo distribuido de una empresa pueden utilizar tecnologías basadas en Grid para alcanzar un recurso compartido que se encuentra más allá de los límites institucionales; en el espacio de ASP/SSP, las tecnologías Grid se pueden utilizar para establecer mercados dinámicos para los recursos de almacenaje y computo, superando las limitaciones debidas a las configuraciones estáticas actuales.

4.3.3.1. Características de las organizaciones virtuales

Para explicar las características de las VOs tomaremos tres organizaciones reales (los óvalos), y dos VOs: P, que liga a participantes en un consorcio aeroespacial del diseño, y Q, que liga a colegas que proporcionaron los ciclos de procesamiento. La organización a la izquierda participa en P, la que esta a la derecha participa en Q, y la tercera a la izquierda es un miembro de P y de Q. (Ver Figura). Las políticas que gobiernan el acceso a los recursos varían según las organizaciones actuales, recursos, y VOs implicadas (ver Figura 5).

Figura 5: Organizaciones participando en distintas VOs compartiendo sus diferentes recursos.



Fuente: The anatomy of the Grid; Ian Foster, Varl Kesselman. Pág 4.

Al compartir un recurso se tiene una condicional: cada dueño del recurso asegura la disponibilidad, conforme a restricciones de cuando, donde, y qué puede ser hecho. Por ejemplo, un participante en la VO P de la figura puede permitir a los socios de la VO invocar su servicio de simulación solamente para los problemas "simples".

Los consumidores de recursos pueden también poner restricciones en las características de los recursos con que están preparados para trabajar. Por ejemplo, un participante en la VO Q pudo aceptar solamente los recursos de cómputo reunidos certificados como "seguros." La puesta en práctica de tales restricciones requiere de mecanismos para expresar políticas de seguridad, establecer la identidad de un consumidor o de un recurso (autentificación), y para determinar si una operación es consistente con las relaciones de compartición aplicables (autorización). Las relaciones para compartir recursos pueden variar dinámicamente en un cierto plazo, en términos de los recursos implicados, debido a la naturaleza del acceso permitido, y debido a los accesos permitidos a los participantes. Estas relaciones no necesariamente implican un sistema explícitamente nombrado de individuos, sino el uso de un diseño multidisciplinario usando programas y datos provenientes de múltiples locaciones y definidos implícitamente por las políticas que gobiernan el acceso a los recursos.

Por ejemplo, una organización puede permitir el acceso a cualquier persona que pueda demostrar que es un cliente o un estudiante. Dada la naturaleza dinámica de las relaciones de compartición de recursos requerimos de mecanismos para descubrir y caracterizar la naturaleza de las relaciones que existan en un punto particular en el tiempo. Por ejemplo, un nuevo participante que se une a la VO Q debe poder determinar a qué recursos puede tener acceso, la calidad de estos recursos, y las políticas que gobiernan el acceso.
Compartiendo relaciones que no son simplemente cliente-servidor, sino peer-to-peer, los proveedores pueden ser a su vez consumidores, y compartir relaciones que puedan existir entre cualquier subconjunto de participantes. Las relaciones

compartidas pueden ser combinadas para coordinar el uso de recursos, perteneciendo cada uno de estos a diversas organizaciones. Por ejemplo, en la VO Q, un proceso que comenzó en un recurso de cómputo puede tener acceso posteriormente a datos o a subcómputos iniciados en otra parte. La capacidad de delegar autoridad de manera controlada llega a ser importante en tales situaciones, al igual que los mecanismos para coordinar operaciones a través de múltiples recursos.

El mismo recurso se puede utilizar de diversas maneras, dependiendo de las restricciones puestas en el uso de los mismos y la meta compartida. Por ejemplo, una computadora se puede utilizar para correr solamente un software específico en un arreglo compartido, mientras que puede proporcionar ciclos genéricos de cálculo en otro. Debido a carencia de conocimiento a priori sobre cómo un recurso puede ser utilizado, limitaciones de funcionamiento, métricas, expectativas, y limitaciones (por ejemplo calidad de servicio) se puede ser parte de las condiciones puestas en el uso del recurso que se comparte.

Estas características y requisitos definen lo que se denomina una organización virtual, un concepto que creemos es fundamental en los sistemas modernos. Las VOs permiten a grupos dispares de organizaciones y/o de individuos compartir recursos de manera controlada, de modo que los miembros puedan colaborar para alcanzar un objetivo compartido.

4.4. Arquitectura de la Grid

La clave en la arquitectura de Grid es la interoperabilidad: el objetivo es establecer la compartición de recursos entre potenciales participantes cualesquiera, lo que se traduce en protocolos comunes. A la hora de especificar la arquitectura, se sigue el modelo de reloj de arena. En esta arquitectura, el cuello de botella, reside en las capas de Recursos y Conectividad, que facilitan la compartición de recursos individuales (ver Figura 6).

Figura 6: Arquitectura de una grilla computacional.

Fuente: Sistemas Distribuidos, María Cruz Valiente Blázquez. Pág. 3.

4.4.1. Infraestructura

Esta capa proporciona los recursos a los que el acceso compartido es conducido por los protocolos Grid (acceso local a recursos lógicos como CPU, software, ficheros), y a la infraestructura compuesta por los recursos computacionales que queremos compartir: Ordenadores individuales, Pool de ordenadores, Clúster, Supercomputadores, Sistemas de almacenamiento.

Es importante que existan mecanismos de interrogación, que permitan descubrimiento de estructura, estado y capacidades, así como mecanismos de gestión que proporcionen control de la calidad del servicio entregada. Los principales elementos que ha de proporcionar son los siguientes:

- Recursos computacionales: para comienzo de programas y control de la ejecución de procesos. Aquí las funciones de interrogación obtienen características de hardware y software e información de estado, y los mecanismos de gestión permiten realizar reserva de recursos. [10]

- Recursos de almacenamiento: obtención y entrega de ficheros. Los mecanismos de gestión permiten controlar recursos para transferencia de ficheros (CPU, discos, ancho de banda), y las funciones de interrogación obtienen información acerca de utilización de recursos, como espacio libre y utilización de ancho de banda. [10]

- Recursos de red: Mediante el uso de mecanismos de interrogación se encamina a obtener las características de la red y su carga, y los mecanismos de gestión proporcionan control sobre recursos de la red (priorización, reserva). [10]

4.4.2. Conectividad

Esta capa define los protocolos de comunicación y autenticación para las transacciones de red Grid. Por lo que se refiere a los requisitos de comunicación, son cubiertos por la pila de protocolos TCP/IP: Internet (IP e ICMP), transporte (TCP y UDP) y aplicación (DNS, OSPF). [10] En cuanto a la autenticación, la capa de conectividad se define con las siguientes características:

- Autenticación de usuario: los usuarios han de identificarse una única vez para acceder a los recursos Grid proporcionados por la capa de infraestructura.

- Delegación: el usuario puede traspasar su acceso a los recursos Grid a un programa.

- Integración con soluciones de seguridad locales: debe existir interoperación con estas soluciones locales, generalmente mediante mapeo en el entorno local de la seguridad Grid.

- Seguridad basada en usuario: si el usuario accede a dos recursos diferentes A y B de manera conjunta, la seguridad de A y B no debe interactuar.

4.4.3. Recurso: Compartir recursos individuales

Está constituida por protocolos que permitirán la negociación segura, iniciación, monitorización, control, pago y tarificación de las operaciones compartidas sobre recursos individuales, mediante la llamada a funciones de la capa de infraestructura para el acceso a los recursos individuales. Notar por tanto que están plenamente enfocados a recursos individuales. Estos protocolos se pueden diferenciar en dos clases:

- De información: obtienen información sobre la estructura y estado de un recurso; por ejemplo configuración, carga actual, costos.

- De control o gestión: se encargan de la negociación del acceso al recurso en términos de requisitos (calidad del servicio, reserva) y operaciones a ser realizadas (creación de procesos, acceso de datos). Son responsables de que la política requerida por el usuario sea consistente con la ofrecida por el recurso.

4.4.4. Recursos: Coordinación de recursos múltiples

Está constituida por los protocolos y servicios no asociados a un recurso específico sino que son globales y capturan interacciones de diferentes recursos. Las principales funciones que lo caracterizan son: [10]

- Servicios de directorio: permiten a las VO participantes descubrir recursos de otras VO.

- Co-ubicación y programación: ubicación de tareas en recursos para un propósito específico y ejecución de tareas en un momento específico.

- Servicios de monitorización y diagnóstico.

- Servicios de descubrimiento de software: elección del mejor software para un determinado problema.

- Gestión de carga de trabajo.

- Servidores de autorización de comunidad.

- Servicios de colaboración.

4.5. Seguridad en la computación Grid

Las necesidades y problemas de seguridad en los sistemas Grid surgen precisamente por las características propias de los entornos de computación Grid:

- La población del usuario es extensa y dinámica. Los participantes de las organizaciones virtuales pueden incluir miembros de diversas instituciones, las cuales, es muy probable que cambien de manera frecuente.

- El servicio de recursos es extenso y dinámico. Debido a que los usuarios y las instituciones individuales son quienes deciden cuando van a contribuir con recursos, la cantidad y la localización de los recursos disponibles puede cambiar de manera muy rápida.

- Una computación (o proceso creado por una computación), durante su ejecución, puede adquirir recursos, comenzar procesos sobre los recursos, y liberar recursos, todo ello de forma dinámica.

- Los recursos pueden requerir diferentes mecanismos y políticas de autenticación y autorización (Contraseñas de texto en claro, Secure Socket Library (SSL)).

- Un usuario individual puede tener asociado diferentes espacios locales de nombres, credenciales, o cuentas, en diferentes sitios, para los propósitos de control de acceso y tarificación.

- Los recursos y los usuarios se pueden localizar en diferentes países.

Como se puede apreciar, se hace necesario proporcionar soluciones de seguridad que permitan la ejecución de procesos en un entorno con las características que se acaban de mencionar. Se deben coordinar las diversas políticas de control de acceso y se debe operar de manera segura en los entornos heterogéneos.

4.5.1. Requerimientos de seguridad

Los sistemas Grid pueden requerir de algunas o todas las funciones estándares de seguridad, incluyendo autenticación, control de acceso, integridad, privacidad y no repudio. Una arquitectura de seguridad puede centrarse principalmente en los aspectos de autenticación y control de acceso: [9]

1. Se deben proporcionar soluciones de autenticación que admiten un usuario, los procesos que constituyen una computación del usuario, y los recursos utilizados por esos procesos, para verificar la identidad de cada uno. La autenticación forma la base de una política de seguridad que permite

diversas políticas de seguridad locales para que sean integradas dentro de un marco global.

2. Se deben permitir mecanismos de control de acceso local para que sean aplicados sin cambios, siempre que sea posible.

4.5.1.1. Restricciones

Para poder desarrollar la arquitectura de seguridad que cumpla los requerimientos anteriores, se deben satisfacer, además, las siguientes restricciones que se derivan de las características del entorno y de las aplicaciones Grid: [9]

- Firma única. Un usuario debería ser capaz de autenticarse una única vez, por ejemplo, cuando se comienza la computación, y poder iniciar computaciones que adquieren recursos, utilizar recursos, liberar recursos, tener comunicaciones internas; todo ello sin necesidad de autenticación por parte del usuario.

- Protección de credenciales. Se deben proteger las credenciales de los usuarios (contraseñas, claves privadas, etc.).

- Interoperabilidad con soluciones de seguridad local. Mientras que las soluciones de seguridad pueden proporcionan mecanismos de acceso entre dominios, el acceso a recursos locales será determinado, de forma típica, por una política de seguridad local que se hace cumplir por un mecanismo de seguridad local. Resulta poco práctico modificar cada recurso local para que se acomode al acceso entre dominios; en vez de eso, una o más entidades en un dominio (ej., los servidores de seguridad entre dominios) deben actuar como agentes de clientes/usuarios remotos para los recursos locales.

- Infraestructura uniforme de credenciales/certificación. El acceso entre dominios requiere, como mínimo, una manera común de expresar la identidad de un principal de seguridad, como sería un usuario o un recurso real.

- Portabilidad. Se requiere que el código sea portable y ejecutable en casos de prueba multinacionales. En resumen, los aspectos de portabilidad indican que la política de seguridad no puede requerir, directa o indirectamente, la utilización de cifrado en masa.

- Soporte para comunicación segura en grupo. Una computación puede estar constituida por un número de procesos que necesitarán coordinar sus actividades como un grupo. La composición de un grupo de procesos puede y de hecho cambiará durante el periodo de vida de un proceso. Por tanto, se necesita soporte para una comunicación segura (en este contexto, autenticada) para los grupos dinámicos.

- Soporte para múltiples implementaciones. La política de seguridad no debería imponer una tecnología de implementación específica. Todo lo contrario, debería ser posible implementar la política de seguridad dentro de un rango de tecnologías de seguridad, basadas todas ellas tanto en la criptografía de clave pública como en la criptografía de clave secreta.

4.5.2. Política de Seguridad

Antes de profundizar en los aspectos específicos de una arquitectura de seguridad, es importante identificar los objetivos de seguridad, las entidades participantes y las suposiciones fundamentales. En resumen, se debe definir una política de seguridad, un conjunto de normas que definan los sujetos de la seguridad (usuarios), objetos de seguridad (recursos) y las relaciones entre ellos. Como son posibles diferentes políticas de seguridad, se presenta una política

específica que trata los aspectos introducidos anteriormente a la vez que refleja las necesidades y las expectativas de las aplicaciones, los usuarios y los propietarios de los recursos.

4.5.2.1. Terminología

Se considera la siguiente terminología de seguridad: [9]

- Sujeto. Un sujeto es un participante en una operación de seguridad. En los sistemas Grid, un sujeto suele ser normalmente un usuario, un proceso que opera en nombre de un usuario, un recurso (ordenador, archivo), o un proceso que actúa en nombre de un recurso.

- Credencial. Una credencial es una porción de información que se utiliza para probar la identidad de un sujeto. Las contraseñas y los certificados son ejemplos de credenciales.

- Autenticación. Autenticación es el proceso por el cual un sujeto prueba su identidad a un interlocutor, normalmente a través de la utilización de su credencial. La autenticación en la cual ambas partes, es decir, los dos interlocutores, se autentican simultáneamente (cada parte se autentica a la otra parte) se conoce como autenticación mutua.

- Objeto. Un objeto es un recurso que está siendo protegido por la política de seguridad.

- Autorización. Autorización es el proceso por el cual se determina si un sujeto tiene permitido el acceso o puede utilizar un objeto.

- Dominio de confianza. Un dominio de confianza es una estructura lógica, administrativa dentro de la cual se mantiene una política de seguridad local,

única y consistente. Dicho de otra manera, un dominio de confianza es una colección de los objetos y de los sujetos gobernados por una administración única y por una política de seguridad única.

4.6. Asignación y planificación de recursos Grid

4.6.1. Servicios para asignación y planificación de recursos Grid

Para hacer más fácil la utilización de las infraestructuras distribuidas que forman el Grid, surge la necesidad de establecer una arquitectura global que sea plasmada en la práctica en una serie de servicios básicos en forma de middleware, y que simplificarán el modo de desarrollar aplicaciones que puedan hacer uso de estas infraestructuras.

En particular un proyecto Grid está orientado principalmente a aplicaciones paralelas e interactivas, por lo que se requiere que el control sobre los recursos y la planificación de los distintos trabajos de estas aplicaciones en concreto sea eficiente, automático y de una manera consistente.

Las tareas relacionadas con la planificación y la selección eficiente de recursos están basadas en un middleware, cuya entidad principal es el Resource Broker (RB), que proporcionará los servicios necesarios para la ejecución automática de trabajos secuenciales.

Mediante estos servicios un usuario es capaz de enviar un trabajo junto con una descripción de los requerimientos del mismo, y el RB se encargará de buscar y seleccionar los recursos necesarios. Cuando se hayan localizado los recursos se pasará a la siguiente etapa que consiste en mandar el trabajo para su ejecución utilizando aquellos seleccionados, tomando las precauciones necesarias y reenviando el trabajo si algo fallara. Continuamente se monitorizará el estado del trabajo para ver que el trabajo se está ejecutando correctamente, reenviando el

trabajo a estos u otros recursos si algo fallara. Se obtendrá finalmente la salida del mismo para reenviarla al usuario, haciendo la ejecución totalmente transparente para éste.

Aunque un proyecto Grid se beneficia de estos servicios, extensiones a éstos y nuevos componentes específicos son necesarias para el control de aplicaciones paralelas interactivas escritas en MPI (Message Passing Interface), y también para solventar algunas deficiencias del sistema original, dando lugar a los Scheduling Agents (SA), que constituyen una extensión de la funcionalidad proporcionada por el RB. [9]

4.6.1.1. Selección de recursos

Una parte importante dentro del sistema de planificación de recursos es poder averiguar cuáles de estos recursos están disponibles en un momento determinado y hacer una selección de los mismos para ejecutar la aplicación.

En la primera fase de la selección el Resource Selector consulta qué recursos cumplen los requerimientos, parciales o totales, que el trabajo está imponiendo. En el caso de se mande un trabajo secuencial éste se deberá ejecutar en un solo CE (Computing Elements / Elementos Computacionales), ya que únicamente requerirá el uso de una CPU que se corresponderá finalmente con un WN (Working Nodes / Nodos Trabajadores) de ese CE. En este caso se construirá una lista de posibles CEs que cumplen todos los requerimientos, ordenados según alguna función de preferencia (número de CPUs del CE libres, potencia de cálculo, memoria libre, etcétera) que el usuario puede establecer.

En el caso de un trabajo MPI se necesitarán varias CPUs para correr el trabajo, que serán seleccionadas en un único CE o en un grupo de ellos, con el siguiente criterio:

- Primero se intentan seleccionar los CEs que tienen todas las CPUs requeridas libres, con lo que se intenta correr primero en un solo clúster para evitar las latencias de los mensajes entre diferentes clúster.

- Después se forman grupos de CEs que cumplan los requisitos necesarios para cada uno de ellos, y que conjuntamente reúnan el número de cpus libres requeridas.

- La ordenación entre los diferentes grupos se hace de menor a mayor número de CEs diferentes, y dentro de los grupos con el mismo número de elementos, se utilizará la función de preferencia ponderada con el número de componentes.

4.6.1.2. Planificación y reservas temporales

Parte de la planificación se ha hecho con el servicio anterior de selección de recursos, ordenando los recursos a utilizar con una serie de criterios que se acaban de nombrar. En la siguiente fase se enviará el trabajo al primer CE o grupo de CEs de la lista en el caso de trabajos MPI. Debido a la naturaleza en la que se obtienen los datos sobre los CEs y sus CPUs libres por ejemplo, que no es completamente actualizada en tiempo real, puede darse el caso de que se produzcan casos en los que el RB/SA considere alguna CPU como libre cuando realmente no lo está porque se acaba de mandar un trabajo. Este es sólo un ejemplo de los casos en los que puede fallar el intento de ejecución de un trabajo, por lo cual es necesario un método para poder reenviarlo a la siguiente selección de la lista.

Además se pueden dar posibles casos de interbloqueos entre diferentes trabajos mandados que requieran varias CPUs, cuando un trabajo se está esperando un WN que está ocupado por un segundo trabajo, y éste está esperando un WN que está ocupado por el primero. Esto se soluciona con un método de reservas locales

temporal que afecta al Resource Selector. Cuando se decide a qué grupo de CEs enviar el trabajo, se establece una reserva por un espacio de tiempo, de manera que estos recursos no serán tenidos en cuenta por el siguiente trabajo que busque recursos.

Estas reservas son locales al SA y no es un sistema general de reservas, sino que simplemente modifica los servicios locales de selección de recursos. Cuando se ha superado cierto intervalo, o el trabajo ha acabado o fallado en esos recursos, la reserva es retirada y esos recursos pueden volver a ser utilizados.

4.6.1.3. Prioridades

Las aplicaciones interactivas requieren un tiempo de respuesta corto para que el usuario pueda estar satisfecho con la ejecución de la misma. Para permitir que este tipo de aplicaciones puedan ejecutarse anteriormente a otras, por ejemplo aplicaciones por lotes, se introduce acá el concepto de prioridades.
Con las prioridades es posible determinar qué trabajos son necesarios ejecutar anteriormente, de manera que si tenemos varios trabajos pendientes de ejecución, podremos determinar el orden en que se van a mandar.

Además en el caso de que todos los recursos estén ocupados por la ejecución de trabajos, y nos llegue uno más prioritario, será posible determinar qué trabajos pueden ser parados momentáneamente para permitir la ejecución del nuevo trabajo más prioritario.

4.6.1.4. Envío del Trabajo

Cuando se tienen todas las decisiones hechas sobre los recursos a los que mandar el trabajo, y las distintas prioridades sobre otros, se deben realizar los pasos adecuados para permitir la ejecución del mismo. Dependiendo del tipo de trabajo, si es por lotes o interactivo, si es secuencial o paralelo escrito en las

diferentes variantes de MPI soportadas, se realizaran diferentes aproximaciones para conseguir el objetivo. El manejador de aplicaciones envía los ejecutables de forma confiable a los distintos recursos seleccionados implicados, y empezar la ejecución. Se pueden Utilizar herramientas como Cóndor-G y ampliaciones al mismo para llevar a cabo estas tareas.

5. MODELO DE DESARROLLO DE APLICACIONES PARA GRID COMPUTING

El desarrollo de aplicaciones en entornos Grid requiere por parte del programador de un conocimiento completo de esta tecnología así como de una gran experiencia en la implementación de aplicaciones para este entorno. Presentaremos un nuevo modelo de desarrollo de aplicaciones en Grid cuyo objetivo es facilitar la implementación de estas aplicaciones ofreciendo además un interfaz de programación estándar. De esta manera, el modelo abstrae al desarrollador de tareas relacionadas con la gestión, mantenimiento y disponibilidad de los recursos conservando el control sobre los trabajos que constituyen la aplicación Grid, además, permite desacoplar las aplicaciones de la infraestructura Grid donde se ejecuten. Para el diseño de un prototipo para manejo de trabajos en el clúster de la UTB utilizaremos Web Services además de la librería *Birdbath* la cual será explicada mas adelante.

Como introducción a los distintos paradigmas de desarrollo en Grid que se utilizan en la actualidad, analizaremos las principales características de estos paradigmas mostrando las ventajas y desventajas de cada una de las soluciones presentadas.

La complejidad intrínseca de las aplicaciones Grid ha facilitado la aparición de herramientas e interfaces que ayudan a los usuarios en su desarrollo y ejecución, de entre todas ellas se pueden encontrar:

- Portales Web: Permiten la ejecución de aplicaciones en entornos Grid a través de servidores Web, un ejemplo de este tipo de interfaces de usuario es el proyecto In-VIGO desarrollado en la Universidad de Florida. In-VIGO permite la ejecución vía Web de aplicaciones en un entorno distribuido formado tanto por recursos físicos como virtuales, de esta manera desvincula a los clientes de las aplicaciones de la complejidad inherente de la computación Grid.

- GRID superscalar: Desarrollado en el Centro Nacional de Supercomputación (BSC-CNS) GRID superscalar se compone de un entorno de programación Grid que permite la ejecución eficiente de programas en Grid computacionales. Para ello, paraleliza en tiempo real y a nivel de tarea aplicaciones secuenciales, generalmente de grano grueso, que posteriormente se ejecutarán en un Grid.

- Interfaces Especificas: Algunos proyectos Grid incorporan dentro de su infraestructura las herramientas necesarias para facilitar el desarrollo de aplicaciones Grid en su marco de ejecución.

Aunque estas herramientas facilitan la ejecución y el desarrollo de aplicaciones Grid bajo determinadas condiciones, lo deseable sería disponer de un modelo general para el desarrollo de dichas aplicaciones. Globus Toolkit, por ejemplo, junto con la herramienta GridWay facilitan la ejecución de trabajos en Grid, el desarrollo de aplicaciones en estos entornos sigue necesitando de esfuerzo adicional por parte del usuario, ya que este se ve en la necesidad de comprender detalladamente el comportamiento del metaplanificador y del middleware a utilizar para que la aplicación llegue a buen término. Para facilitar estas labores al desarrollador el paradigma o modelo de desarrollo debe ofrecer las siguientes características:

- Transparencia: El modelo de desarrollo debe abstraer al desarrollador de todas las tareas relacionadas con la gestión de los recursos del Grid, así como de las tareas vinculadas con la planificación de los trabajos que se ejecuten. Sin embargo, el desarrollador debe tener en todo momento control sobre los trabajos en ejecución para así poder comprobar y modificar el estado de los mismos.

- Fiabilidad: Si durante la ejecución de la aplicación un recurso falla, el modelo debe permitir que la ejecución continúe enviando, por ejemplo, los trabajos asignados a ese recurso a otro que se encuentre disponible. Sea cual sea la tarea que se lleve a cabo, esta debe ser transparente tanto desde el punto de vista del desarrollador como del usuario de la aplicación.

- Adaptación Dinámica. Los trabajos pueden migrar a los recursos que mejor se adapten a sus necesidades durante la ejecución de la aplicación. Además, si se incorpora un nuevo recurso al Grid, este podrá ser utilizado durante el resto de la ejecución de la aplicación

Todas estas características permiten desacoplar las aplicaciones Grid de la infraestructura Grid a la que se encuentre asociada

5.1. MPICH-G2

MPICH-G2 es una versión adaptada a entornos Grid de una implementación del estándar MPI denominada MPICH. El estándar MPI (*Message Passing Interface*) es una especificación basada en el paso de mensajes realizada por consenso por el MPI Forum para los lenguajes de programación C, C++ y Fortran. Concretamente, MPI especifica un modelo de programación para desarrollar aplicaciones que pueden ser utilizadas en sistemas de SPMD (*Single Program Multiple Data*). Siguiendo este modelo, un mismo programa se ejecuta en todos los nodos del sistema, mientras que los datos son distribuidos equitativamente entre cada uno de los nodos, de tal forma que cada nodo procesa solo aquellos datos que le corresponden. De esta manera, la comunicación entre los distintos procesos, recuérdese que cada proceso ejecuta el mismo programa, se realiza a través de llamadas a funciones de envío y recepción de mensajes.

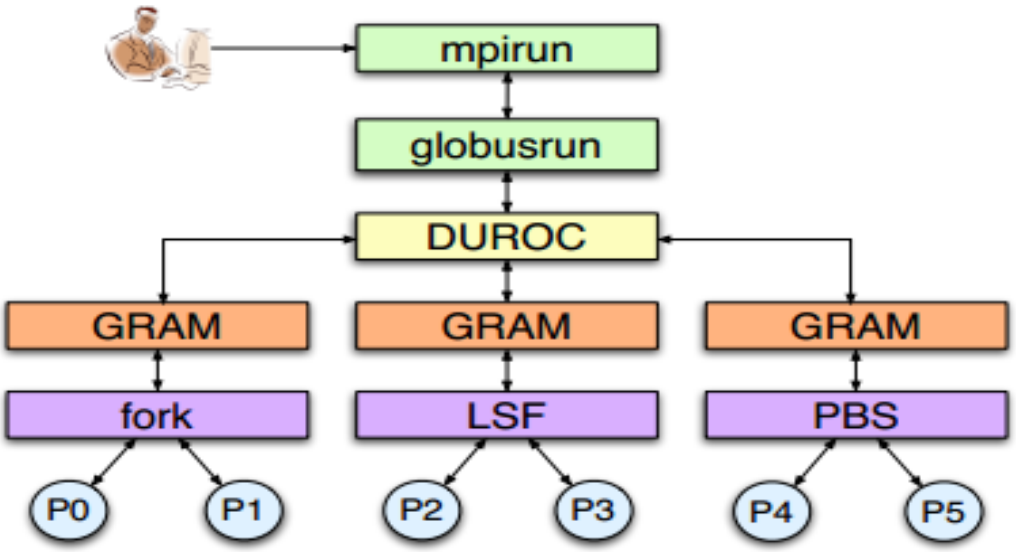
El estándar MPI ha ido evolucionando desde la versión MPI 1.0 publicada en el año 1994, hasta su versión más reciente MPI 2.0, que incorpora extensiones para

la ejecución de trabajos según el modelo MPMD (*Multiple Program Multiple Data*), permitiendo a cada proceso ejecutar un programa distinto.

Aunque está orientado preferiblemente a sistemas homogéneos, MPI permite exportar sus aplicaciones a sistemas multiprocesadores, multicomputadores o sistemas heterogéneos. Además, al ser un estándar basado en funciones facilita la portabilidad y el desarrollo de aplicaciones, ya que el usuario desarrolla la aplicación paralela como si de un programa secuencial se tratase incorporando las funciones MPI que sean necesarias. Existen diversas implementaciones del estándar MPI, como LAM/MPI o MPICH. Es importante destacar, que ninguna de estas implementaciones está orientada a entornos Grid, sino que son implementaciones que siguen estrictamente el modelo de paso de mensajes especificado en el estándar MPI.

MPICH-G2 es la implementación del estándar MPI, concretamente de la versión MPI 1.1, para entornos Grid. Dicha implementación utiliza servicios de la herramienta Globus para realizar el paso de mensajes entre los diferentes nodos. El paradigma de programación MPICH-G2 es una versión renovada de MPICH-G, y permite la ejecución de programas desarrollados bajo la plataforma MPI aprovechando todos los servicios y facilidades que ofrece la herramienta Globus, como pueden ser gestión de trabajos, gestión de seguridad y servicios de conversión de datos. Básicamente MPICHG2 ofrece un interfaz que realiza la paralelización de la aplicación, convirtiendo sus datos en mensajes que permitirán la comunicación entre sistemas con distintas arquitecturas.

Figura 7: Arquitectura general de MPICH-G2



Fuente: Modelo de Programación para Infraestructuras Grid Computacionales, José Herrera Sanz, Madrid 2008, Pág. 31.

Inicialmente, el usuario realiza un programa, por ejemplo en lenguaje C, al que incorpora las funciones definidas en las librerías de MPICH-G2. Seguidamente, ejecuta el comando **mpirun** que realiza las tareas necesarias para permitir la ejecución del programa utilizando el paradigma MPICH-G2 como por ejemplo describir el trabajo a ejecutar utilizando el lenguaje RSL (*Resource Specification Language*). Con toda la información contenida en el fichero RSL, MPICH-G2 utiliza la librería DUROC (*Dinamically Updated Request Online Coallocator*) para planificar y ejecutar los trabajos en los recursos Grid especificados en dicho fichero. Si dichos nodos se encuentran disponibles DUROC utilizará GRAM para enviar los trabajos a los LRMS de cada nodo, en caso contrario se debe modificar el fichero RSL para consultar la disponibilidad de otros nodos del Grid.

La figura 8 presenta un programa en MPICH-G2 que implementa una aplicación sencilla maestro-esclavo. Dicha aplicación realiza una comunicación en anillo

entre un nodo maestro y un conjunto de nodos esclavos especificados en la llamada a dicha aplicación a través del comando **mpirun**.

Figura 8: Ejemplo de un programa básico en MPICH-G2

```
#include <mpi.h>

int main(int argc, char **argv)
{
    int numprocs, my_id, passed_num, trip;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_id);

    if (numprocs > 1)
    {
        if (my_id == 0)  /* Proceso Maestro */
        {
            passed_num = 1;
            MPI_Send(&passed_num, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
            MPI_Recv(&passed_num, 1, MPI_INT, numprocs-1, 0, MPI_COMM_WORLD, &status);
            printf("Maestro: fin del viaje: después de recibir %d pasajes", passed_num);
        }
        else            /* Proceso Esclavo */
        {
           MPI_Recv(&passed_num, 1, MPI_INT, my_id-1, 0, MPI_COMM_WORLD, &status);
           passed_num ++;
           MPI_Send(&passed_num, 1, MPI_INT, (my_id+1)%numprocs, 0, MPI_COMM_WORLD);
        } /* endif */
    }
    MPI_Finalize();
    exit(0);
} /* end main() */
```

Fuente: Modelo de Programación para Infraestructuras Grid Computacionales, José Herrera Sanz, Madrid 2008, Pág. 32.

El objetivo de la aplicación consiste en contabilizar el número total de pasajes de un determinado viaje. Para ello, cada uno de los nodos incrementa en una unidad la variable *passed_num* encargada de almacenar el número total de pasajes. Como se puede observar, la comunicación entre los distintos procesos se realiza a través de envío y recepción de mensajes utilizando las llamadas *MPI_Send* y *MPI_Receive*. Además, las llamadas *MPI_Comm_size* y *MPI_Comm_rank* permiten al desarrollador conocer tanto el número de procesos que se van a utilizar como el identificador único de cada uno de estos procesos.

El paradigma MPICH-G2 se ha utilizado para el desarrollo de aplicaciones distribuidas en diversos proyectos, sin embargo no incluye todas las características necesarias para su consideración como modelo de programación en entornos Grid. Ya que un modelo de programación Grid debería incorporar características como transparencia, fiabilidad y adaptación dinámica. Aunque MPICH-G2 facilita la labor al desarrollador gracias al uso del comando **mpirun** y la librería DUROC, la trasparencia que ofrece este paradigma no es suficiente ya que el usuario debe controlar los recursos donde se realizará la ejecución de la aplicación. Además, debido al uso de la barrera DUROC, es necesario que todos los recursos especificados se encuentren disponibles durante la ejecución de la aplicación para que esta se realice con éxito.

MPICH-G2 no permite la adaptación dinámica de las aplicaciones y por lo tanto desaprovecha las capacidades computacionales de los nuevos recursos que se incorporen al Grid una vez iniciada la ejecución de la aplicación. Tampoco se ajusta a los patrones típicos de aplicaciones Grid, como en el caso de aplicaciones de alta productividad o de flujo de trabajo, incrementando además el tiempo de ejecución de las aplicaciones ya que el estándar MPI está pensado para el desarrollo de aplicaciones en sistemas fuertemente acoplados, y por lo tanto no tiene en cuenta las latencias que se producen en las redes públicas. Finalmente, es importante destacar que MPICH-G2 realiza la comunicación directa con Globus, por lo que desaprovecha todas las facilidades y funcionalidades que ofrecen los metaplanificadores.

5.2. SAGA

El paradigma de programación SAGA (*Simple API for Grid Applications*) es una especificación para aplicaciones Grid definida por OGF (Open Grid Forum), que surge con el objetivo de simplificar el desarrollo de aplicaciones en Grid. Para ello, proporciona un API simple que permite estandarizar la sintaxis y semántica de dichas aplicaciones. Además, el interfaz propuesto por SAGA permite el desarrollo

de aplicaciones independientes del middleware utilizado. Las principales características del API SAGA son:

- Simplicidad. SAGA es fácil de usar, instalar, administrar y mantener.

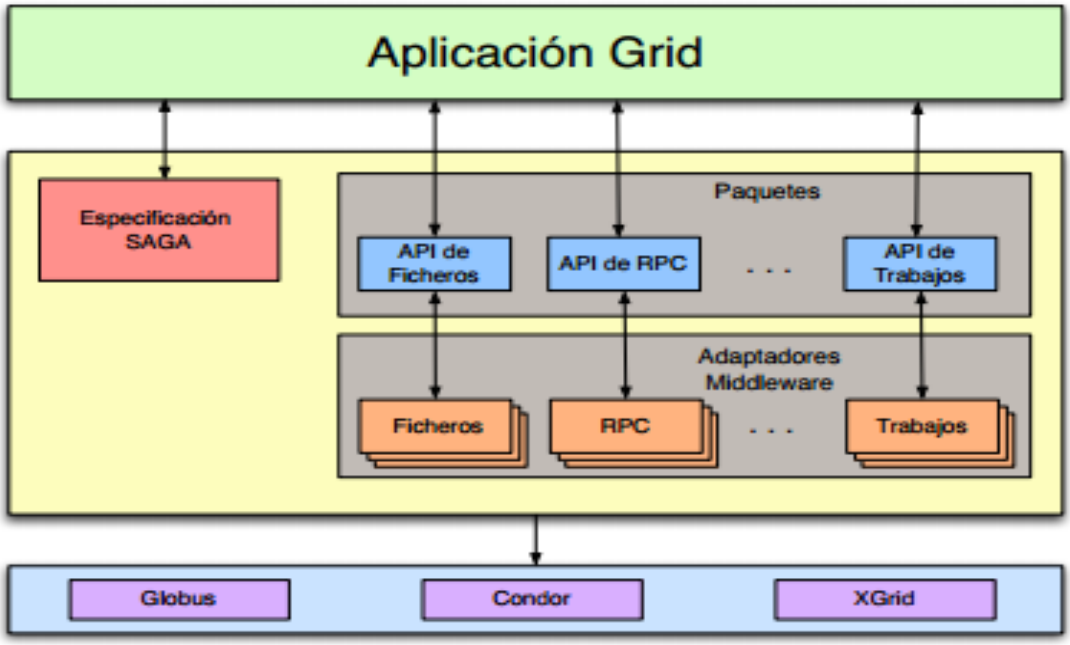
- Uniformidad. SAGA ofrece soporte a diversos lenguajes de programación como C++ o Java, así como una semántica y estilo conforme a las diferentes funcionalidades del Grid.

- Escalabilidad. Contiene mecanismos que permiten la ejecución de una misma aplicación en una gran variedad de sistemas, añadiendo soporte para distintos middleware.

- Modularidad. El API SAGA ofrece un entorno de trabajo fácilmente extensible por parte del desarrollador de aplicaciones.

La figura 9 muestra un esquema general de la arquitectura SAGA. Como se puede comprobar, la arquitectura está compuesta por varios módulos:

Figura 9: Arquitectura General de SAGA



Fuente: Modelo de Programación para Infraestructuras Grid Computacionales, José Herrera Sanz, Madrid 2008, Pág. 34.

- Especificación SAGA. Contiene todo el motor necesario para que el desarrollador implemente una aplicación Grid utilizando las sentencias proporcionadas por el API SAGA.

- Paquetes SAGA. Proporcionan sentencias especificas para la gestión de ficheros, RPC, trabajos, espacio de nombres y flujo de datos.

- Adaptadores Middleware. Se encargan de realizar la traducción de las sentencias proporcionadas por el módulo denominado “Paquetes SAGA” a sentencias especificas del middleware.

De entre todos los módulos que componen la arquitectura SAGA el más importante es el correspondiente al módulo denominado “Especificación SAGA”, ya que constituye el eje central de este paradigma de programación. A

continuación se detallan los elementos más importantes que componen dicho módulo:

- Gestor de Espacio de Nombres. Se encarga de gestionar el espacio de nombres de cada uno de los ficheros físicos y directorios que componen la aplicación que se está desarrollando siguiendo el estándar POSIX. Define, por tanto, una estructura jerárquica compuesta por directorios y entradas, donde cada directorio puede contener a su vez otro directorios o entradas, y cada entrada puede contener un objeto SAGA. El gestor de espacio de nombres ofrece sentencias para manejar enlaces simbólicos, lista de control de acceso, búsquedas de directorios y entradas, y operaciones basadas en patrones.

- Gestor de Ficheros. Permiten la manipulación y control de los ficheros a través de sentencias SAGA basándose en el estándar POSIX. Estas sentencias pueden leer, escribir y realizar búsquedas sobre los ficheros físicos, independiente del middleware utilizado.

- Gestor de Trabajos. Realiza la gestión de trabajos a través de sentencias basadas en el API DRMAA, pero sin completar todos los servicios que ofrece este API. Por tanto el desarrollador puede crear trabajos, describir cada uno de los elementos del trabajo (definidos en el estándar JSDL) y gestionar el estado de cada trabajo.

- Gestor de Replicas. Ofrece servicios de replicas de ficheros y directorios similar a Globus RSL, añadiendo a las llamadas ofrecidas por el gestor de espacio de nombres llamadas para la gestión de replicas.

- Gestor de Flujo de Datos. Realiza la gestión de conexión de sockets similar a los sockets BSD.

Además, SAGA también ofrece llamadas para el control de sesiones dentro de la aplicaciones Grid. Permitiendo de esta manera agrupar conjuntos independientes de objetos SAGA dentro de una misma sesión. Finalmente la especificación SAGA se realiza a través del lenguaje SIDL (*Scientific Interface Definition Language*), existiendo en la actualidad versiones de SAGA para los lenguajes C++ y Java que utilizan dicha especificación.

La figura 10 presenta un ejemplo sencillo de un programa en C++ con la especificación SAGA. Dicho programa realiza dos funciones básicas, en primer lugar prepara y envía al Grid un programa simple para su ejecución y en segundo lugar espera por la finalización de su ejecución. Como se puede observar, la espera que realiza es de las denominadas esperas activas, es decir, el programa pregunta constantemente por el estado del trabajo en ejecución hasta que este termina de ejecutarse, esto es debido a que el API de SAGA no incluye un comando que espere por la finalización de los trabajos.

Figura 10: Ejemplo de un programa básico en SAGA

```
#include <string>
#include <saga/saga.hpp>
int main(int argc, char **argv)
{
    saga::job_description jobdef;
    jobdef.set_attribute("Executable", "job.sh");
    saga::job_service js;
    saga::job job = js.create_job("remote.host.net", jobdef);
    job.run();
    while( job.get_state() == saga::job::Running )
    {
        std::cout << "Job running with ID: "
        << job.get_attribute("JobID") << std::endl;
        sleep(1);
    }
}
```

Fuente: Modelo de Programación para Infraestructuras Grid Computacionales, José Herrera Sanz, Madrid 2008, Pág. 36.

Aunque el API SAGA se utiliza en diversos proyectos ofreciendo muchas ventajas en el desarrollo de aplicaciones en sistemas Grid, el comportamiento que presenta

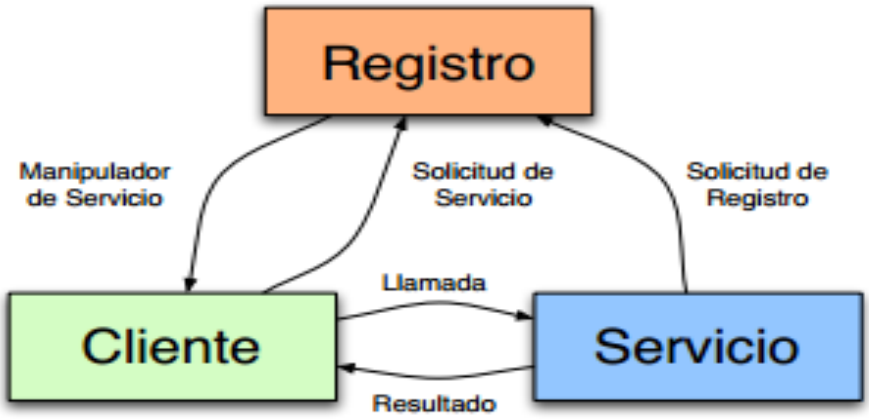
es muy similar al de un pseudometaplanificador. De hecho, SAGA se encuentra en una capa intermedia entre el middleware, en el caso de la presente tesis sería Globus, y el usuario. Por tanto al igual que ocurre con MPICH, obvia todas las facilidades y servicios que ofrecen los metaplanificadores presentados en el capítulo anterior. De hecho los autores comparan en varias ocasiones SAGA con MPI definiendolo como: “SAGA es el homologo Grid de MPI ".

5.3. Grid-RPC

Grid-RPC constituye un paradigma de programación compuesto por un mecanismo de llamadas a procedimientos remotos (RPC, Remote Procedure Calls) para entornos Grid.

Grid-RPC permite el desarrollo de aplicaciones en Grid a través de la comunicación entre el usuario y el middleware utilizando llamadas a procedimientos remotos. De esta manera, abstrae al programador de las tareas de comunicación entre los elementos dinámicos del Grid así como de la gestión de seguridad que todo entorno Grid requiere.

Figura 11: Modelo General de Grid-RPC



Fuente: Modelo de Programación para Infraestructuras Grid Computacionales, José Herrera Sanz, Madrid 2008, Pág. 37.

En primer lugar, un servicio determinado solicita su registro, enviando su información al registro de servicios. Posteriormente, una vez que el servicio está registrado, el cliente puede solicitar dicho servicio al registro de servicios. De esta manera, el cliente recibirá el manipulador del servicio especifico, que le permite realizar la llamada al procedimiento remoto. Una vez realizada la llamada al servicio, el cliente recibe el resultado de dicha llamada. Como se puede comprobar este comportamiento es muy similar a los que existen en otras arquitecturas como ocurre en el caso de CORBA. Es importante añadir, que el servicio puede ser un servicio web si el cliente está tratando con el middleware Globus. Los aspectos más importantes del paradigma Grid-RPC son:

- Manipulador de Funciones e Identificadores de Sesión. El manipulador de funciones se encarga de obtener la instancia concreta de una función de un determinado servicio a partir del nombre de dicha función. Por otro lado, el identificador de la sesión representa a una determinada llamada RPC no bloqueante.

- Funciones de Inicio y Finalización. Las llamadas realizadas antes de la inicialización o después de la finalización no serán válidas.

- Funciones de Gestión de Funciones Remotas. Permite la creación y destrucción de manipuladores de función remotos.

- Funciones de Llamadas GridRPC. Permiten realizar llamadas a procedimientos remotos, tanto de forma bloqueante como no bloqueante, así como con distintos argumentos de entrada.

- Funciones de Espera y Control. Permiten comprobar el estado de las solicitudes así como esperar a la finalización de las llamadas no bloqueantes previamente enviadas.

- Funciones de Informes de Error. Ofrecen una descripción de los errores que se hayan podido producir durante la ejecución.

- Funciones de Pila de Argumentos. Permiten construir los argumentos de una llamada en tiempo de ejecución.

Estas funciones facilitan la gestión de llamadas a procedimientos remotos a través del Grid, permitiendo de esta manera que la aplicación cliente tenga un aspecto similar a la de una aplicación RPC clásica.

Figura 12: Ejemplo de un programa básico en Grid-RPC

```
#include <string>
#include <grpc.h>
int main(int argc, char **argv)
{
    grpc_function_handle_t handle;
    grpc_sessionid_t id;
    int i, time, ret, a, b, result;
    time = 1;
    a=4;
    b=5;

    if (ret = grpc_initialize(argv[1])) ==  GRPC_NO_ERROR)
    {
        grpc_function_handle_init(&handle,"magi1.cpe.ku.ac.th",
        "/example/plus");

        if ((grpc_call_async(&handle, &id, a, b, &result) &&
             grpc_wait_all() == GRPC_NO_ERROR)
          grpc_function_handle_destruct(&handles[i]);

        grpc_finalize();
    }
    exit(0)
}
```

Fuente: Modelo de Programación para Infraestructuras Grid Computacionales, José Herrera Sanz, Madrid 2008, Pág. 38.

Este programa ejecuta en un nodo remoto una instancia de un procedimiento previamente implementado denominado “plus” que realiza la suma de dos números, para a continuación esperar por la finalización de dicho procedimiento. Para ello el nodo destino debe tener el ejecutable de dicho programa, ya que

desde el cliente se realizará una llamada asíncrona a dicho procedimiento de forma remota. Como se puede observar, Grid-RPC utiliza llamadas para realizar ejecución remota de procedimientos, dificultando de esta manera el desarrollo de aplicaciones, ya que el proceso de paralelización de dichas aplicaciones se realiza a través de la distribución de procedimientos y funciones en lugar de distribuir trabajos completos. Además, debido a que dichos procedimientos deben estar implementados en el nodo destino, el desarrollador debe conocer de antemano los nodos que contienen dichos métodos. Por lo tanto el paradigma Grid-RPC carece de características fundamentales (dinamismo, transparencia, fiabilidad y adaptación dinámica) que faciliten y abstraigan al desarrollador de las tareas de comunicación, planificación o gestión de recursos.

Con el objetivo de incrementar el dinamismo de las aplicaciones Grid-RPC han surgido diversos proyectos que desarrollan implementaciones de dicho paradigma como NetSolve y Ninf-G. Estas implementaciones ofrecen al programador infraestructuras para el desarrollo y despliegue de funciones y procedimientos remotos, facilitando al desarrollador la gestión de dichas funciones y procedimientos en los nodos remotos a través del uso de stubs. Aunque estas implementaciones ha sido utilizadas en variedad de proyectos de investigación como, por ejemplo, el estudio de simulaciones climatológicas, estas mejoras no logran suplir las deficiencias inherentes al propio paradigma, ya que, como se ha comentado previamente, no lo abstraen al desarrollador de las tareas relacionadas con la gestión de recursos.

5.4. DRMAA

DRMAA (*Distributed Resource Management Application API*) es una especificación desarrollada por el grupo de trabajo DRMAA-WG perteneciente al Global Grid Forum (*GGF*). Esta especificación es utilizada para la ejecución y el control de los trabajos en uno o varios DRMS (*Distributed Resource Management Systems*). El ámbito de esta especificación consiste en todas aquellas funcionalidades de alto nivel que son necesarias para que una determinada

aplicación pueda enviar trabajos a sistemas DRM incluyendo operaciones sobre dicho trabajos como por ejemplo, operaciones de finalización, suspensión y reinicio. El objetivo de este especificación es facilitar una interfaz de aplicación directa entre los sistemas DRM actuales y los programadores de aplicaciones, portales o los distribuidores de software independiente.

5.4.1. Aportación del modelo DRMAA

La especificación DRMAA constituye un interfaz homogéneo y portable a diferentes DRMS para gestionar el envío, monitorización y control de trabajos, y para la recuperación del estado de los trabajos finalizados. Las implementaciones y las aplicaciones distribuidas de DRMAA, no tienen que particularizarse para un entorno DRMS determinado, o para una políticas de desarrollo determinadas. Para ello se utiliza las categorías de trabajos y la especificación nativa del DRMAA. De esta manera, las políticas especificas de la aplicación se pueden traducir en simples cadenas de caracteres que interpretarán las implementaciones que utilicen el API DRMAA.

DRMAA ofrece interfaces de “categorías de trabajos” que encapsulan los detalles concretos del recurso donde se ejecutarán los trabajos, ocultando dichos detalles a las aplicaciones que utilicen la interfaz DRMAA. De esta forma, el administrador del recurso, puede crear una categoría de trabajo, sujeta a una determinada aplicación que se ejecutará usando DRMS. El nombre de dicha categoría se especifica como un atributo de ejecución del trabajo. De esta forma, la implementación DRMAA, puede usar ese nombre de la categoría para manejar los recursos específicos y los requisitos funcionales de los trabajos pertenecientes a esa categoría.

Por lo tanto, el concepto de categorías ofrece ocultación de las políticas determinadas de cada recurso a los trabajos que se estén ejecutando usando el API DRMAA. Por otro lado para facilitar la implementación de dichas categorías, se debe mantener una categoría por cada política del recurso utilizada. De esta forma, la especificación nativa de las categorías, permite interpretar cada política a

través de una simple cadena de caracteres, que será interpretado por cada una de las librerías del API DRMAA.

De esta manera, el API DRMAA permite el desarrollo de aplicaciones en Grid ocultado al usuario cualquier detalle relativo al gestor de recursos distribuidos o al middleware utilizado. Por lo tanto, el desarrollador realizará un programa de forma secuencial, con sentencias muy parecidas al POSIX de Unix, y DRMAA, el metaplanificador o gestor de recursos distribuidos, y el middleware se encargarán del resto.

En contrapartida, los resultados parciales de las ejecuciones deberán ser almacenados en ficheros ubicados en memoria secundaria aumentado en determinadas ocasiones, el tiempo total de ejecución. [11]

5.4.2. Especificación del modelo

Existen diversas especificaciones del API para diferentes lenguajes de programación, en esta tesis se detallarán los aspectos más importantes de la especificación del estándar para el lenguaje C#. Desde el punto de vista de este lenguaje, la especificación DRMAA se compone de métodos y clases. El objetivo de esta sección consiste en presentar una visión global de la especificación DRMAA, por ello únicamente se comentarán los aspectos más generales de esta especificación, indicando las funciones o métodos más importantes de dicha API.

Antes de especificar las funciones más importantes que ofrece el estándar DRMAA, es importante explicar el concepto de sesión. Toda implementación de una aplicación que se realice bajo el API DRMAA debe estar dentro de una sesión DRMAA. Es decir, cualquier llamada a un método o función del API que no se encuentre dentro de una sesión, previamente iniciada, no será tratada y se enviará un mensaje de error. Por lo tanto, todas las llamadas DRMAA deberán estar ubicadas entre la llamadas de inicio y de fin de sesión. El motivo de la existencia de sesiones es permitir la independencia de los elementos que se encuentren dentro de una misma sesión del resto de elementos semejantes que se

encuentren en ejecución. Como se puede observar el concepto de sesión DRMAA es muy similar al concepto de sesión en SAGA.
La especificación del API DRMAA se divide en cuatro grandes grupos de métodos (clases) o funciones, para unificar conceptos, se denominarán tanto a los métodos como a la funciones, llamadas, ya que el objetivo es presentar una estructura general de las funcionalidades del API, y no una descripción de cada implementación realizada. Por lo tanto el estandar API se divide en cuatro grupos de llamadas:

- Llamadas de Sesión. Corresponden a las llamadas encargadas de inicializar y finalizar la sesión DRMAA. La llamada encargada de iniciar la sesión, inicializa las estructuras de datos necesarias en una sesión de DRMAA, mientras que la llamada encargada de finalizar la sesión libera las estructuras de datos, dejando la aplicación como se encontraba antes del inicio de la sesión. [11]

- Llamadas de Definición de Trabajos. Permiten la manipulación de entidades de definición, es decir, plantillas de trabajos. De esta forma, permite establecer parámetros tales como el fichero ejecutable, sus argumentos, flujos de salida estándar, variable de entorno y directorio de trabajo local. DRMAA permite extender estas llamadas a llamadas propias y especificas del gestor de recursos a utilizar.
  Dependiendo del lenguaje de programación estas llamadas pueden ser un conjunto de definiciones, funciones y constantes declaradas en un módulo, o pueden constituir una clase base a partir de la cual se pueden extender todas las clases hijas que se deseen. [11]

- Llamadas de Ejecución de trabajos. Estas llamadas son el eje fundamental dentro de la API DRMAA, se encargan de enviar los trabajos para su ejecución. Permiten el envío de un trabajo independiente, así como el envío de un conjunto de trabajos. Tanto en un caso como en el otro, el API

DRMAA devuelve un identificador único que representa al trabajo enviado al Grid para su ejecución dentro de la sesión DRMAA. [11]

- Llamadas de Control y Monitorización de Trabajos. En este grupo se engloban todas aquellas llamadas que se utilizan para controlar y sincronizar trabajos así como para monitorizar su estado. Las llamadas de control permiten parar la ejecución, reenviar o matar tanto un trabajo específico como la totalidad de trabajos que estén en ejecución en una determinada sesión. Por otro lado, las llamadas de sincronización permiten esperar a la finalización de ejecución de un trabajo determinado, de un conjunto de trabajos o de todos los trabajos de la sesión. Finalmente, las llamadas de monitorización devuelven el estado actual de un determinado trabajo. [11]

Esta especificación permite el desarrollo de programas a través de un estándar facilitando así el despliegue de aplicaciones en Grid. Además, el uso del API DRMAA permite desarrollar aplicaciones Grid independientemente del gestor de recursos utilizado. También proporciona al desarrollador llamadas para gestión de los trabajos en ejecución, posibilitando de esta manera la implementación de aplicaciones compatibles con los perfiles de ejecución típicos de Grid, como aplicaciones de alta productividad o aplicaciones maestro-esclavo.[11] Y lo más importante, facilita el desarrollo de aplicaciones que se desacoplen de la infraestructura Grid donde se vayan a ejecutar.

5.5. BirdBath

La tecnología de Servicios Web se ha convertido en un elemento importante en el diseño y desarrollo de una infraestructura grid. La capacidad de descomponer los recursos y la funcionalidad que ofrecen en un conjunto de servicios de detección y ligeramente acopladas aborda muchos de los problemas de interoperabilidad que se pueden encontrar en las grandes infraestructuras de escala de la grilla. A través

de la conformidad con un conjunto establecido de normas basadas en XML, como SOAP (*Simple Object Access Protocol*) para la comunicación y WSDL (*Web Service Definition Language*) para la definición de la interfaz, los servicios web garantizan que las aplicaciones son desarrolladas de forma independiente y las herramientas pueden ser plenamente interoperables .
Actualmente llevar en línea el middleware Grid y sus herramientas con esta evolución es una gran oportunidad para identificar nuevas formas de interactuar con estas tecnologías y aprovechar más las capacidades de los recursos que gestionan.

Condor es un muy buen candidato para tal tarea. Se trata de una programación de trabajos ampliamente adoptado y su sistema de gestión de los recursos ofrece una amplia gama de servicios bien definidos de computación de alto rendimiento. Al exponer su funcionalidad como un conjunto bien definido de servicios web individuales, permiten que los recursos Condor se logren integrar perfectamente en un entorno de servicios emergente orientado a la autorización a terceros para incorporar capacidades de Condor en sus propias aplicaciones, y mejorar considerablemente la habilidad del Condor para operar a través de fronteras organizacionales.

Condor ofrece una gama rica y variada de servicios, que se puede simplificar en las tres categorías siguientes:

- Planificación de tareas: Condor proporciona un medio para gestionar las solicitudes de trabajo de ejecución de las colas constantes de trabajos, así como coordinar y supervisar la ejecución remota de los puestos de trabajo. [12]

- Servicios de gestión de recursos: un gestor central es responsable de recoger las características del recurso y de información sobre el uso de las máquinas en una plataforma Condor. Se basa en información recopilada, y

en las prioridades del usuario, las solicitudes del trabajo se pueden adecuar a los recursos necesarios para su ejecución. [12]

- Gestión de trabajos de ejecución: La central gestiona la ejecución remota de trabajos en los recursos seleccionados. Condor ofrece la posibilidad guardar trabajos asegurándose de que pueden ser trasladados a otros recursos en caso de fallo. También proporciona la capacidad de redirigir las llamadas al sistema de presentación de la máquina, así como la funcionalidad de transferencia de archivos hacia y desde el lugar de la ejecución. [12]

Toda estas características y funcionalidades mencionadas se encuentran encapsuladas en una librería de clases llamada BirdBath. La gestión de trabajos, manejo de sesiones, listado de colas y todas las funcionalidades existen como métodos lo cual nos permite su fácil manejo. Utilizando los dos principales web services proporcionados por Condor: Condor_Scheduller y Condor_Collector realizamos la interacción con el clúster de la UTB permitiéndonos el envió y monitoreo de trabajo. Para ampliar información sobre BirdBath ver referencia [12].

5.6. Diferencias entre arquitecturas para desarrollo en computación Grid

Hasta hace poco tiempo se había enfocado el problema de la programación grid sobre los modelos tradicionales de programación paralela, sin embargo, es relevante considerar que puede ser insuficiente aplicar solo éstas técnicas para los retos de la computación grid.

Los estándares de programación grid deben entonces permitir crear aplicaciones que cumplan con requerimientos tales como: usabilidad, soporte para ambientes heterogéneos, portabilidad, interoperabilidad, confidencialidad y seguridad, tolerancia a fallos, entre otros, un evento tan simple como el "acceso al estado" de un recurso puede influir significativamente sobre el modelo de programación.
Debido a la necesidad de cumplir con estas características se han creado diferentes paradigmas como modelos a seguir para el desarrollo de aplicaciones grid:

MODELOS	CARACTERISTICAS PRINCIPALES
Orientado a Memoria Compartida (OpenMP)	• Permite la ejecución paralela de código, la definición de datos compartidos y la sincronización de procesos.
Orientado a Memoria Distribuida (MPI)	• Permite acoplar múltiples máquinas, potencialmente con diferentes arquitecturas para correr aplicaciones. • Mantiene activos todos los procesos durante la ejecución de la aplicación.
Peer To Peer	• Escalabilidad: Las redes P2P tienen un alcance mundial con cientos de millones de usuarios potenciales. En general, lo deseable es que cuantos más nodos estén conectados a una red

	P2P, mejor será su funcionamiento. • Robustez. La naturaleza distribuida de las redes peer-to-peer también incrementa la robustez en caso de haber fallos en la réplica excesiva de los datos hacia múltiples destinos. • Descentralización. • Anonimato. • Seguridad.
RPC	• Servidor independiente • Orientada al proceso y el hilo modelos orientados apoyados por RPC. • El desarrollo de sistemas distribuidos es simple, ya que utiliza la semántica sencillas y fácil. • Al igual que las comunicaciones comunes entre las partes de una aplicación, el desarrollo de los procedimientos para las llamadas remotas es bastante general. • El procedimiento requiere conserva la lógica de negocio, que es apto para la aplicación. • Se minimiza el código de re-escritura / re-desarrollo de esfuerzo. • Permite el uso de las aplicaciones utilizadas en el entorno distribuido, no sólo en el entorno local.
Basada en componentes y objetos	Las principales alternativas que se utilizan para la programación distribuida han sido hasta el momento: • Sockets: Uno de los inconvenientes es que requiere una implementación costosa, en términos de usar APIS complejas y tediosas para el programador. • Remote Procedure Calls (RPC): No soporta

	objetos explícitamente. • Microsoft Distributed Component Object Model (DCOM): Menos maduro, menos portable y además propietario. • Java Remote Method Invocation (RMI): Debe ser Java – To – Java. • Common Object Request Broker Architecture (CORBA): Multiplataforma, multilenguaje, restricciones en los puertos de comunicación. • Web Services: Multiplataforma, multilenguaje. No tiene restricciones de puertos por usar servicios sobre protocolos http estándar.
Orientado a servicios	Los modelos orientados a servicios están escalando a través del estándar OGSA (Open Grid Services Architecture), este modelo presenta una arquitectura basada en tecnologías de Web Services, soportada en estándares como XML, WSDL, SOAP, UDDI. Su arquitectura está basada en Web Services (WSDL)

Arquitectura	**Modelo**	**Ventajas**	**Desventajas**
MPICH-G2	MPI	Provee un simple y bien definido paradigma de programación basado en un especifico paso de mensajes y comunicación colectiva.	• No tiene transparencia. • Carece de fiablidad. • No tiene adaptación dinámica. • Realiza la comunicación directamente con Globus desaprovechando

			todas las facilidades y funcionalidades que ofrecen los metaplanificadores.
SAGA	RPC	• Simplicidad • Uniformidad • Escalabilidad • Modularidad	• Realiza la comunicación directamente con Globus desaprovechando todas las facilidades y funcionalidades que ofrecen los metaplanificadores.
GRID-RPC	RPC	• No tiene que preocuparse de conseguir una dirección de transporte único (un socket en una máquina). • El servidor puede unirse a cualquier puerto y registrar el puerto con su servidor de nombres RPC. • El cliente se pondrá en contacto este servidor de nombres y solicitar el número de puerto que se	• Carece de dinamismo. • No es transparente. • No es fiable. • Carece de adaptación dinámica. • Dificulta el desarrollo por sus llamadas para ejecutar remotamente los procedimientos.

		corresponde con el programa que necesita. • El sistema es independiente del transporte. Esto hace que el código sea más portátil para entornos que pueden tener diferentes proveedores de transporte en uso. • También permite que los procesos en un servidor en estar disponibles a través de todos los proveedores de transporte en un sistema. • Las aplicaciones en el cliente sólo necesita saber una dirección de transporte: la del proceso del servidor de nombres. • No se necesita saber el número de puerto de cada proceso de servidor que quieren ponerse en contacto.	

DRMAA	RPC	• Oculta del usuario cualquier detalle al gestor de recursos o el middleware utilizado. • Maneja sesiones obligatoriamente. • Permite desarrollar aplicaciones independientemente del gestor de recursos utilizados.	• Carece de dinamismo. • No es transparente. • No es fiable. • Carece de adaptación dinámica.
BIRDBATH	Orientado a servicios	• Establecimiento de identidad y negociación de autenticaciones • Descubrimiento, monitorización y gestión de servicios • Negociación y monitorización de niveles de servicio Comunicación y Gestión de la Virtualización de miembros de la Grid. • Uso jerárquico de servicios Grid.	• Gastos generales: La transmisión de datos en XML no es tan conveniente como códigos binarios.

		• Integración de recursos de datos en los procesos de computación. • Gestión de recursos a través de plataformas heterogéneas. • Generación de Calidad de Servicio (QoS) adecuada. • Definición de una base común para gestión autónoma (capacidad de los sistemas de auto verificarse y autogestionarse) • Interfaces públicos y abiertos. • Utilización de tecnología estándar: SOAP; XML, … • Integración con los recursos de TI existentes.	

6. ESPECIFICACION DE LA PROBLEMATICA

Cuando el término GRID apareció en la década de los noventa, no se tenía la menor idea de su definición exacta ni de su posible verdadero uso, por lo tanto se emprendió la tarea de aclarar esta nube y así dar a entender que en esencia el GRID proporciona colaboración e integración entre estas tecnologías, así como la creación de nuevas API (Application Programming Interfaces) y SDK (Software Development Kits).

Primero se debe mencionar que la tecnología GRID es muy diferente de las tecnologías que normalmente se conocen, ya que estas sólo están enfocadas a ser utilizadas por un tipo determinado de arquitectura y forma de implementación, es decir, son usadas por computadores que funcionan conjuntamente pero dada la manera como se configuraron en la red y las limitaciones tecnologías sólo funcionan para esa infraestructura y no pueden integrarse coordinadamente con otros sitios que presenten características totalmente distintas.

Por el contrario, el GRID no se refiere a un único tipo de servicio que se pone en marcha en una determinada red, precisamente es todo lo contrario, provee la integración de diferentes servicios y redes para la obtención de un bien común.

Este tipo de integración entre redes conformó uno de los mayores desafíos para el GRID, donde para referirse a esta situación se acuñó el término INTEROPERABILIDAD. Este término paradójicamente trae a colación uno de los problemas técnicos más grandes que tuvo que enfrentar el GRID, pero también una de sus mayores fortalezas, debido a que deber ser completamente abierto y permitir manejar cualquier tipo de infraestructura que posea, no existe software que permitan manejar la grilla computacional de forma sencilla, intuitiva y amigable, lo cual es la necesidad por la que se realiza esta tesis, delimitando y enfocando nuestra área de trabajo en el cluster que se encuentra en la UTB.

6.1. Solución del problema

Teniendo en cuenta la necesidad de la UTB de ofrecerle a los usuarios finales (estudiantes, docentes, investigadores, etc) una aplicación que les permita hacer uso de la grilla de la universidad de una manera rápida, sencilla, ágil pero sobre todo amigable, hemos creado este diseño del prototipo de GUI para el manejo de la misma.

Es claro que para obtener una mejora en la manipulación de los trabajos dentro del clúster de la UTB, los procesos realizado por los usuarios deben enviarse por medio de una plataforma amigable, que requiera solo pasos sencillos como el login del usuario, crear el trabajo, enviarlo, editarlo en caso dado, y poder realizar un listado de los mismos, para llevar un control de avance de cada uno de los trabajos en los cuales está el clúster actuando.

El objetivo fundamental de este software es generar un ambiente amigable entre el usuario final y el clúster realizador de los trabajos. Dentro de las características previstas de esta aplicación, se proponen las siguientes:

1. Un proceso de validación de usuario, donde el aplicativo valida los datos desde un formulario. El sistema recibe los datos y realiza las validaciones necesarias, y es aquí donde devuelve un mensaje de error si no se logró pasar las validaciones correspondientes, si es en caso contrario se retorna un mensaje de éxito.
2. Un servicio de edición de perfil de usuario, que posee como pre-condición que el usuario este creado y habilitado en el clúster de la UTB, y donde el fin del servicio es la edición del perfil de usuario, la cual se hace por medio de un formulario donde se ingresan los datos de usuario y se envían, el sistema valida los datos y procede a actualizar el perfil del usuario.
3. La función principal Crear Trabajos, esta se inicia cuando el usuario hace clic en el menú Trabajos, submenú Crear. El sistema desplegara una

ventana donde el usuario debe especificar las propiedades del trabajo que desea crear.

4. Creado los trabajos, y estando habilitados en el clúster, tenemos la opción de editarlos con la función existente llamada Editar Trabajos. La que permite abrir un trabajo y modificar sus propiedades.
5. Función enviar, que inicia cuando el usuario hace clic en el menú Trabajos, submenú Enviar. El sistema desplegara una ventana con el listado de todos los trabajos creados, el usuario debe seleccionar un trabajo para enviarlo a ser procesado.
6. Listar trabajos, en este caso se hace clic en el menú Trabajos, submenú Listado. El sistema desplegara una el listado de todos los trabajos enviados a procesar así como el estado en que se encuentran, también se podrían listar los trabajos que existen en cola, es decir , los trabajos que no pudieron ser enviados al clúster.

6.2. Impacto esperado

El impacto esperado de este diseño de prototipo de GUI esta relacionado primordialmente con generar una interacción y uso más constante del clúster de la UTB, por medio de la utilización de la aplicación y con brindar respuestas más optimas en relación con los trabajos en proceso de ejecución del clúster. Algunos de los beneficios se pueden detallar a continuación:

- Una interacción mucho más amigable con el clúster de la UTB, donde el usuario puede ver detalladamente las propiedades de determinado trabajo.
- Información en tiempo real sobre el estado de un determinado trabajo, por quien fue enviado, que tiempo de ejecución lleva.
- Listado de todos los trabajos realizados por el clúster, en realización y en espera a ser ejecutados.
- Un mejor manejo y control sobre el uso del clúster, usuarios que envían trabajos, tipos de trabajos que envían y tiempo de respuesta del clúster.

6.3. Diseño

El crecimiento permanente en computación paralela y sus arquitecturas en hardware y software, han generado nuevos retos para los desarrolladores de software. La computación Grid cada vez toma más relevancia en los problemas que requieren soluciones de multiprocesamiento y grandes cargas de trabajo; en consecuencia los modelos de programación deben adaptarse a estos desarrollos para que aprovechen y soporten las características de paralelismo, distribución, transparencia, heterogeneidad, rendimiento y procesamiento.

Como previamente se menciona el componente principal para el diseño del prototipo de GUI para monitoreo de trabajos del cluster de procesamiento de la UTB es la librería BirdBath, basándonos en el cuadro comparativo creado anteriormente (Ver capitulo *5.6. Diferencias entre arquitecturas para desarrollo en computación Grid*) utilizaremos dicha librería para la interacción del prototipo con el cluster mediante el uso de los web services que provee.

La librería contiene un conjunto de clases y métodos que permiten realizar todo lo necesario para poder enviar un trabajo al cluster y poder monitorear su procesamiento. Permite realizar funciones desde autenticación de usuarios hasta listado de trabajos que están por procesar. *Ver Anexo A, Anexo B.*

El prototipo consta de funcionalidades y requerimientos que permitirán ejecutar todo tipo de procesos en cuanto a envío de trabajos se refiere. En este capítulo se indicara como está conformado el prototipo y cuales serán sus principales funciones.

6.3.1. Especificaciones funcionales

Las especificaciones funcionales del prototipo de GUI para monitoreo de trabajos del cluster de procesamiento de la UTB son las listadas a continuación.

ID REQUERIMIENTO: RF001
NOMBRE: Editar Perfil de Usuario
DESCRIPCION: El sistema permitirá al usuario registrado previamente en el clúster de la UTB editar sus datos básicos.
TERMINOS: • **Datos básicos:** Nombre, Apellido.
PRIORIDAD: Alta (X) Media () Baja ()
AUTOR: Álvaro García.

ID REQUERIMIENTO: RF002
NOMBRE: Crear trabajo
DESCRIPCION: El sistema permitirá al usuario crear un trabajo desde el aplicativo, donde le permita asignar las propiedades del trabajo como tal.
TERMINOS: • **Propiedades del trabajo:** Universo, recurso grid, archivo ejecutable, argumentos, output, archivo de error, archivo de log.
PRIORIDAD: Alta (X) Media () Baja ()
AUTOR: Álvaro García.

ID REQUERIMIENTO: RF003
NOMBRE: Editar trabajo
DESCRIPCION: El sistema permitirá al usuario editar un trabajo desde el aplicativo, donde le permita modificar las propiedades del trabajo como tal.
TERMINOS: • **Propiedades del trabajo:** Universo, recurso grid, archivo ejecutable, argumentos, output, archivo de error, archivo de log.
PRIORIDAD: Alta (X) Media () Baja ()
AUTOR: Álvaro García.

ID REQUERIMIENTO: RF004
NOMBRE: Enviar trabajo
DESCRIPCION: El sistema permitirá al usuario enviar un trabajo al cluster de la UTB para que pueda ser procesado.
TERMINOS: • **Propiedades del trabajo:** Universo, recurso grid, archivo ejecutable, argumentos, output, archivo de error, archivo de log.
PRIORIDAD: Alta (X) Media () Baja ()
AUTOR: Álvaro García.

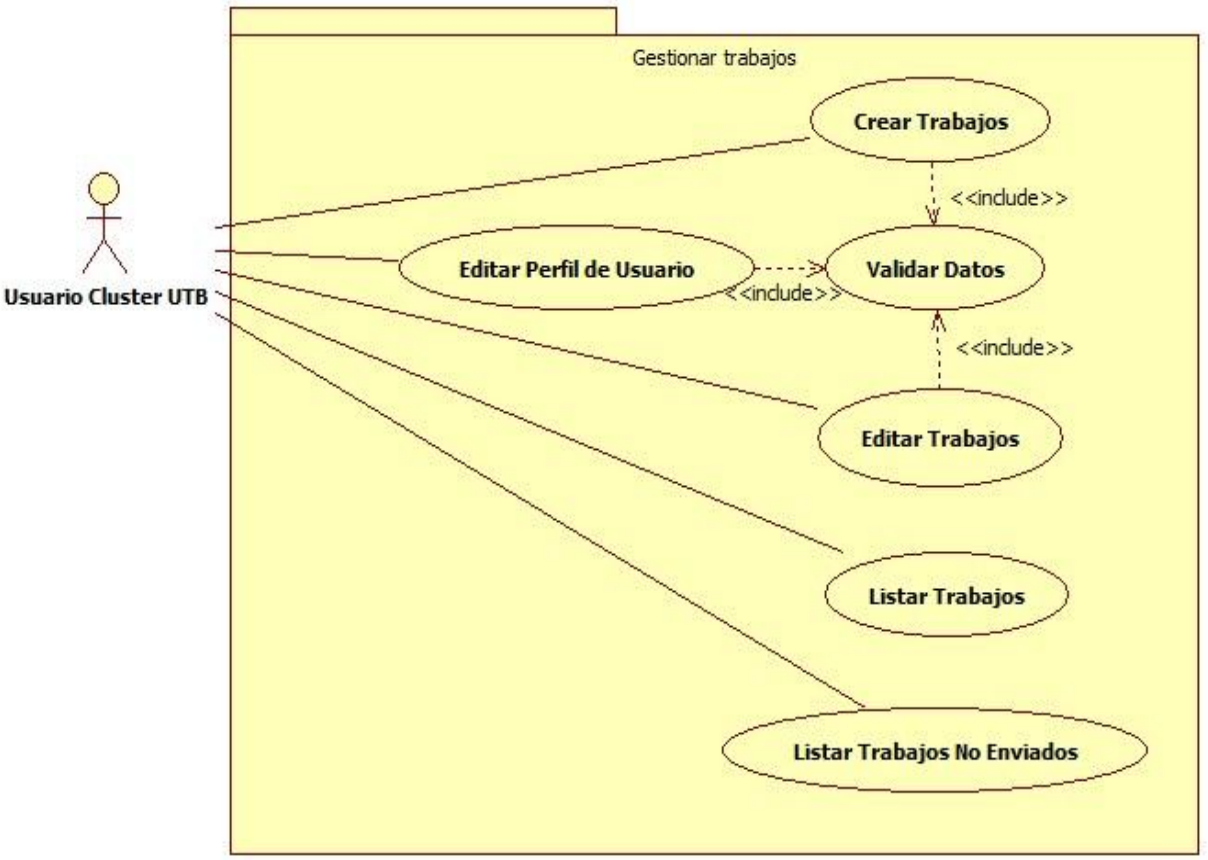
ID REQUERIMIENTO: RF005
NOMBRE: Listar trabajos
DESCRIPCION: El sistema permitirá al usuario listar los trabajos que ha enviado para saber el estado en que se encuentran.
TERMINOS: • **Propiedades del trabajo:** Universo, recurso grid, archivo ejecutable, argumentos, output, archivo de error, archivo de log.
PRIORIDAD: Alta (X) Media () Baja ()
AUTOR: Álvaro García.

ID REQUERIMIENTO: RF006
NOMBRE: Listar trabajos no enviados.
DESCRIPCION: El sistema permitirá al usuario listar los trabajos que no se han podido enviar indicando el motivo por el cual no lo ha hecho.
TERMINOS: • **Propiedades del trabajo:** Universo, recurso grid, archivo ejecutable, argumentos, output, archivo de error, archivo de log.
PRIORIDAD: Alta (X) Media () Baja ()
AUTOR: Álvaro García.

6.3.2. Casos de uso

Los casos de uso del prototipo de GUI para monitoreo de trabajos del cluster de procesamiento de la UTB son los listados a continuación:

- Diagrama Gestionar trabajos



- Validar login: El sistema debe ser capaz de validar las credenciales de acceso.

- Editar Perfil de Usuario: El sistema permitirá al usuario registrado previamente en el clúster de la UTB editar sus datos básicos.

- Crear trabajo: El sistema permitirá al usuario crear un trabajo desde el aplicativo, donde le permita asignar las propiedades del trabajo como tal.

- Editar trabajo: El sistema permitirá al usuario editar un trabajo desde el aplicativo, donde le permita modificar las propiedades del trabajo como tal.

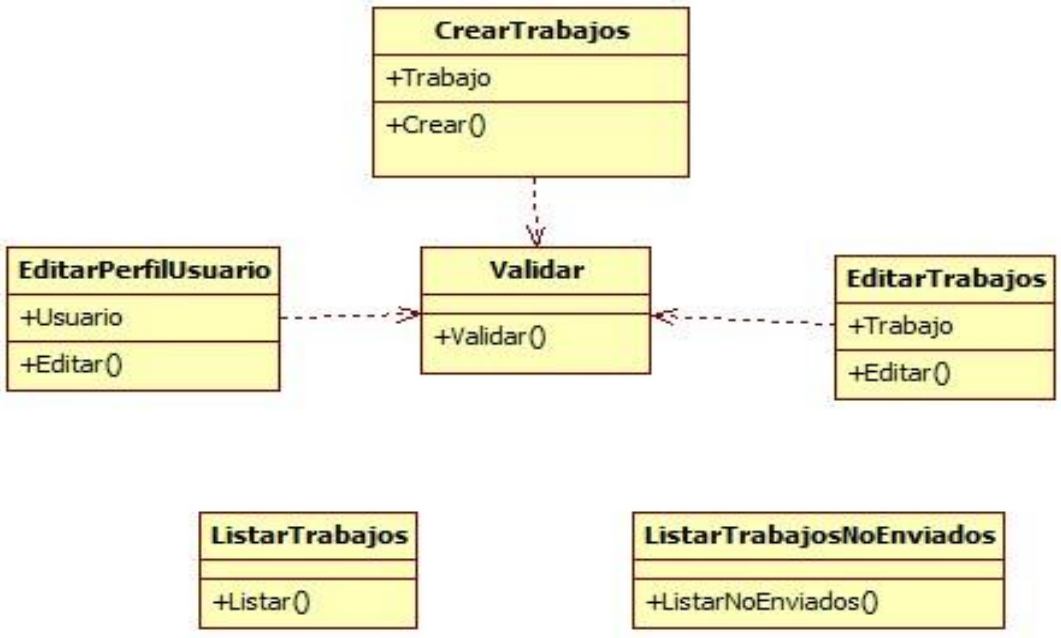
- Enviar trabajo: El sistema permitirá al usuario enviar un trabajo al cluster de la UTB para que pueda ser procesado.

- Listar trabajos: El sistema permitirá al usuario listar los trabajos que ha enviado para saber el estado en que se encuentran.

- Listar trabajos no enviados: El sistema permitirá al usuario listar los trabajos que no se han podido enviar indicando el motivo por el cual no lo ha hecho.

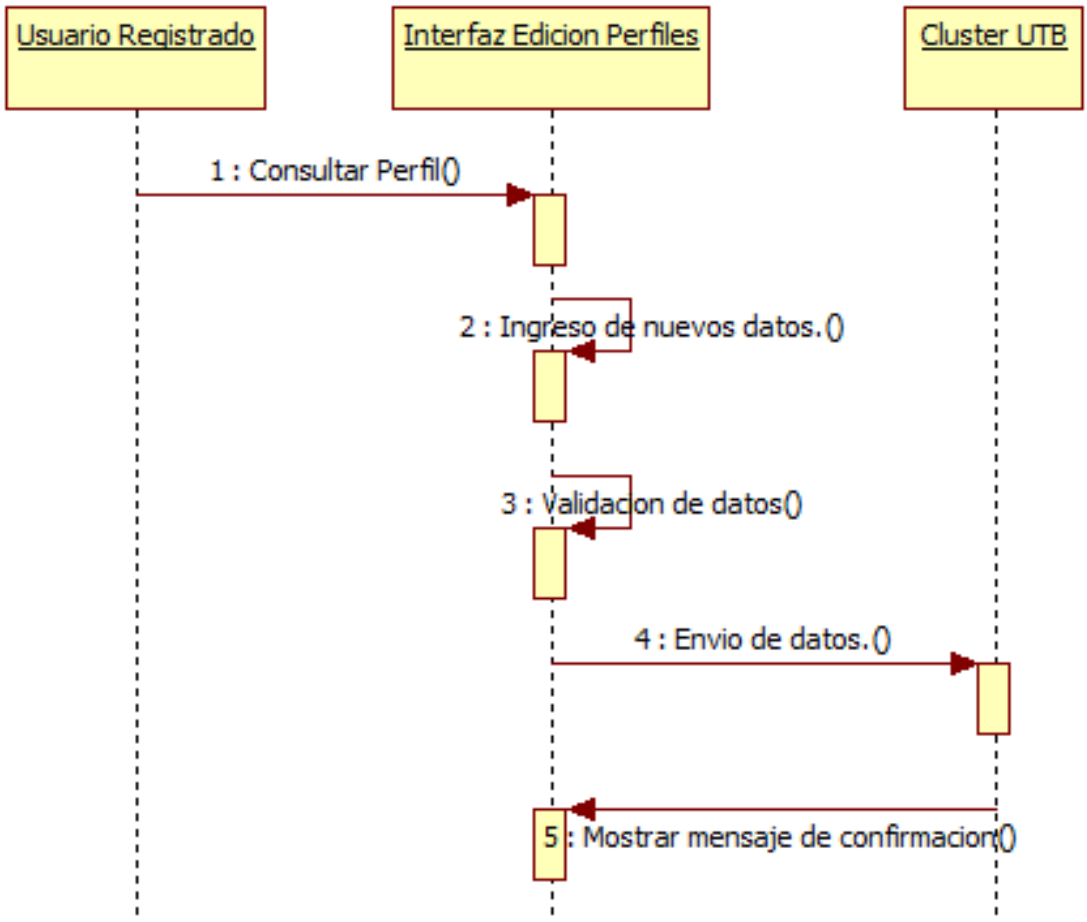
6.3.3. Diagramas de clases

Clases del prototipo para el monitoreo de trabajos del cluster de procesamiento de la UTB:

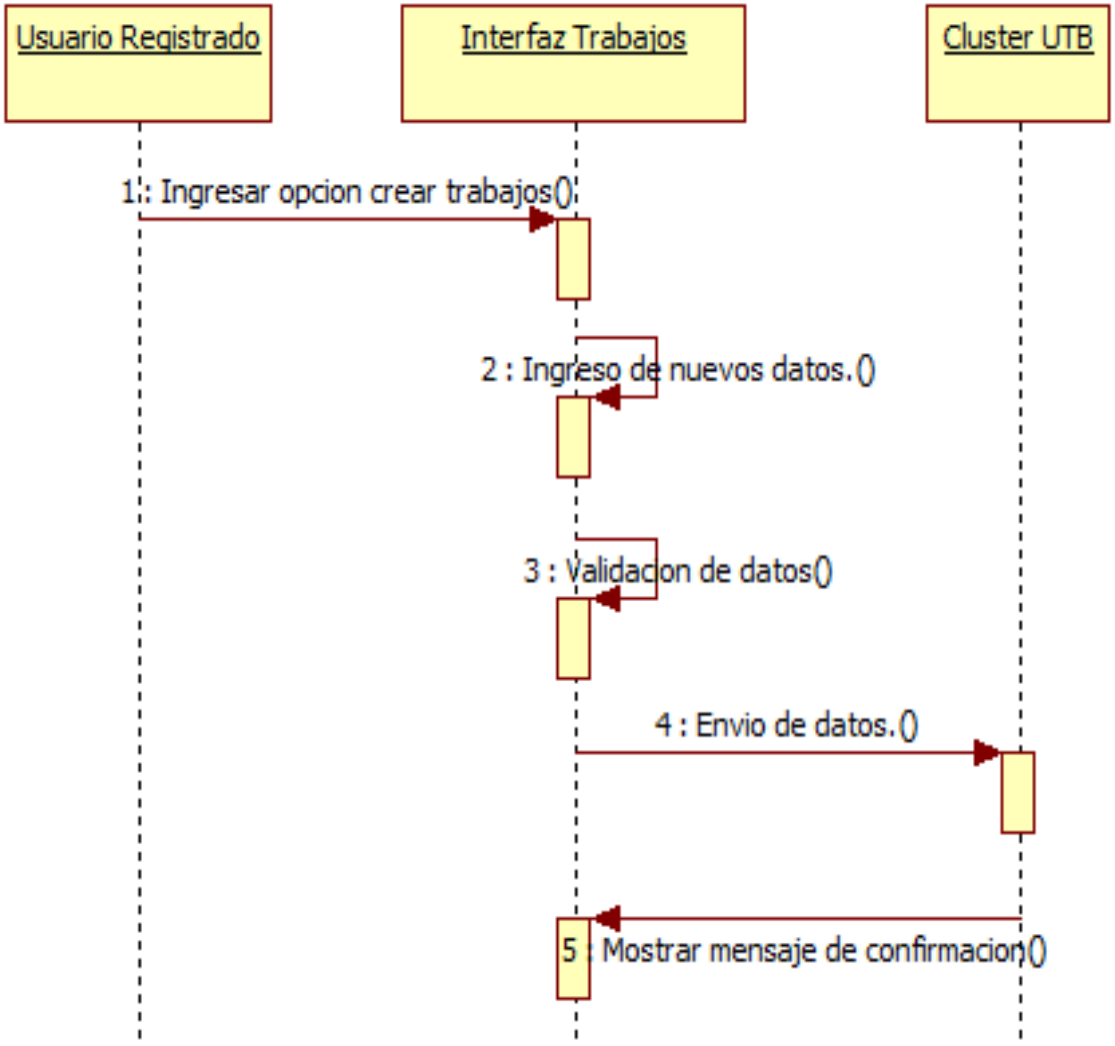


6.3.4. Diagramas de secuencia

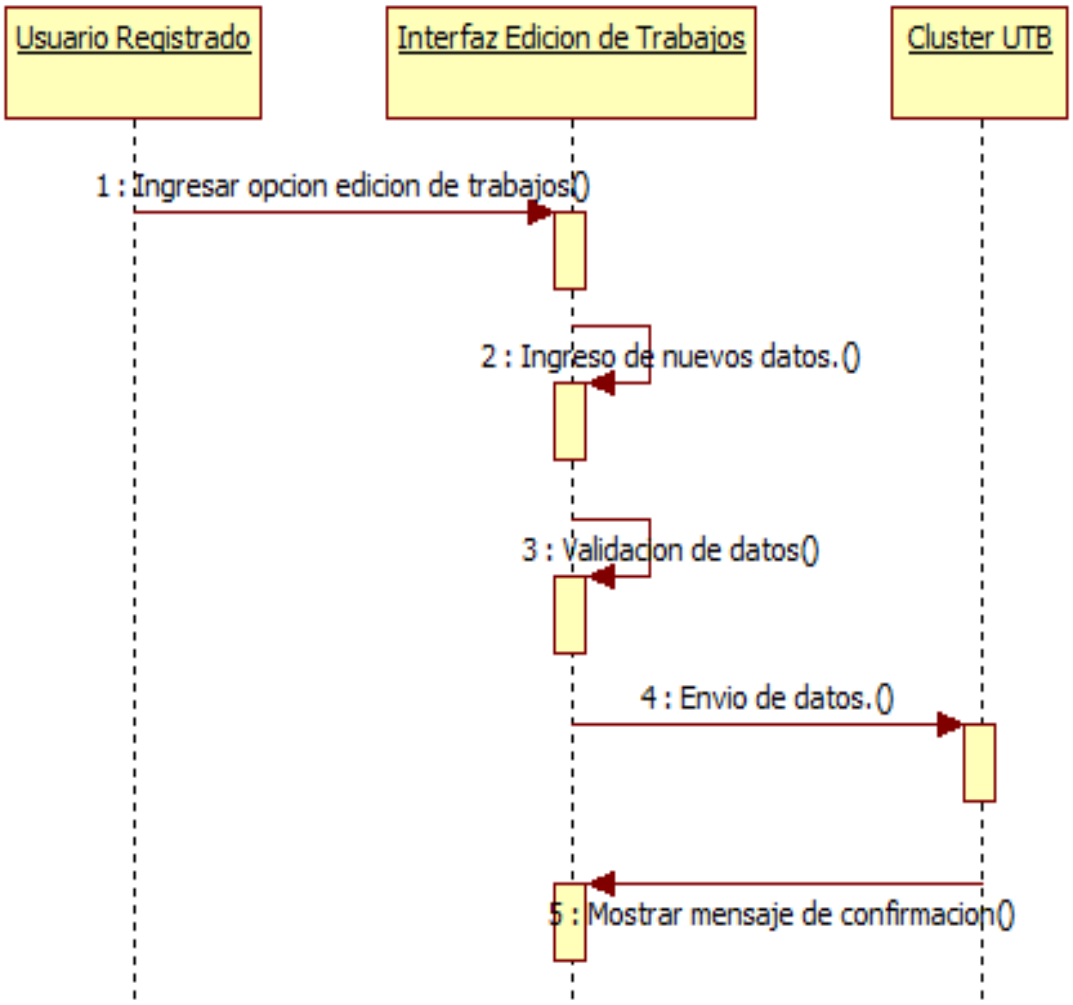
- Editar perfil de usuario



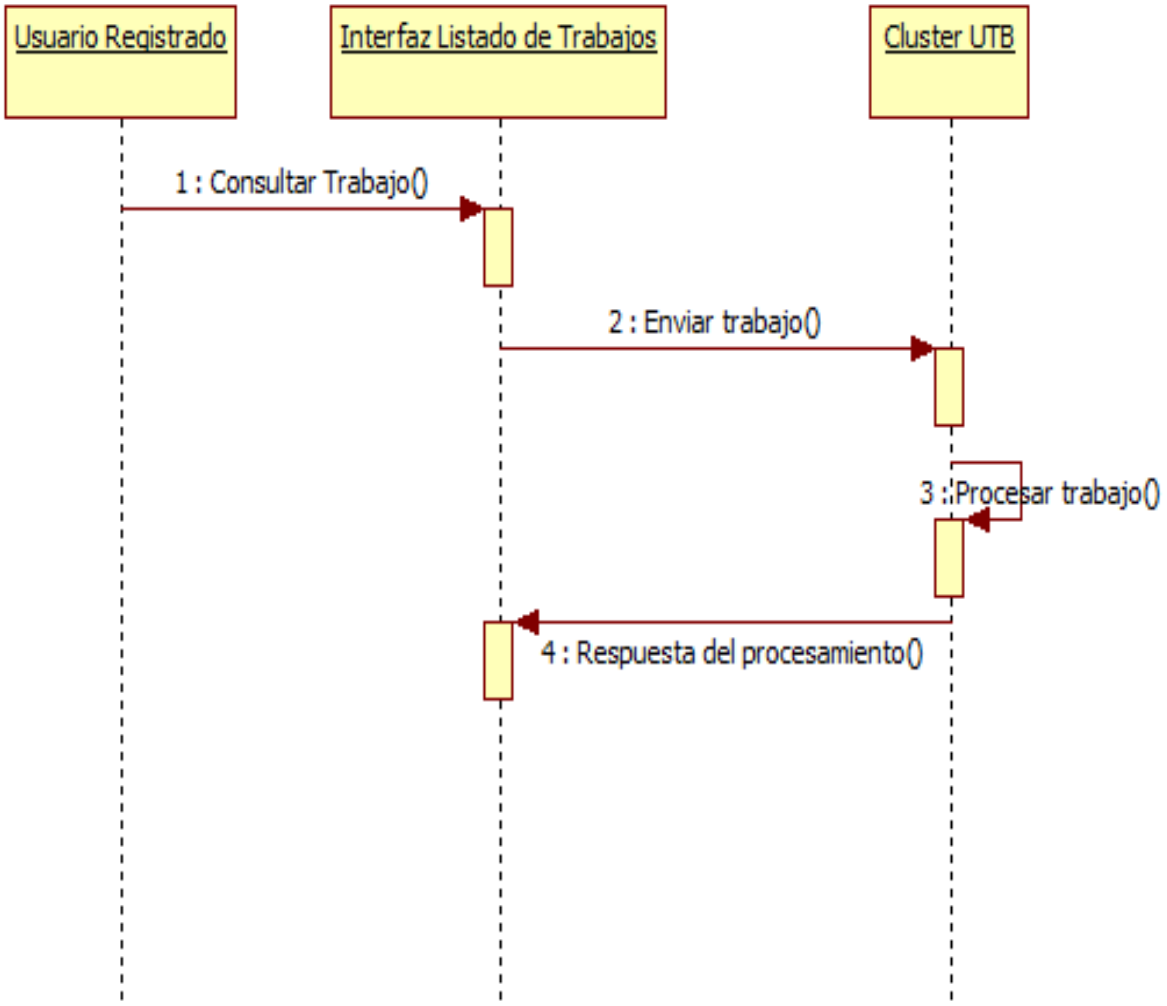
- Crear trabajos



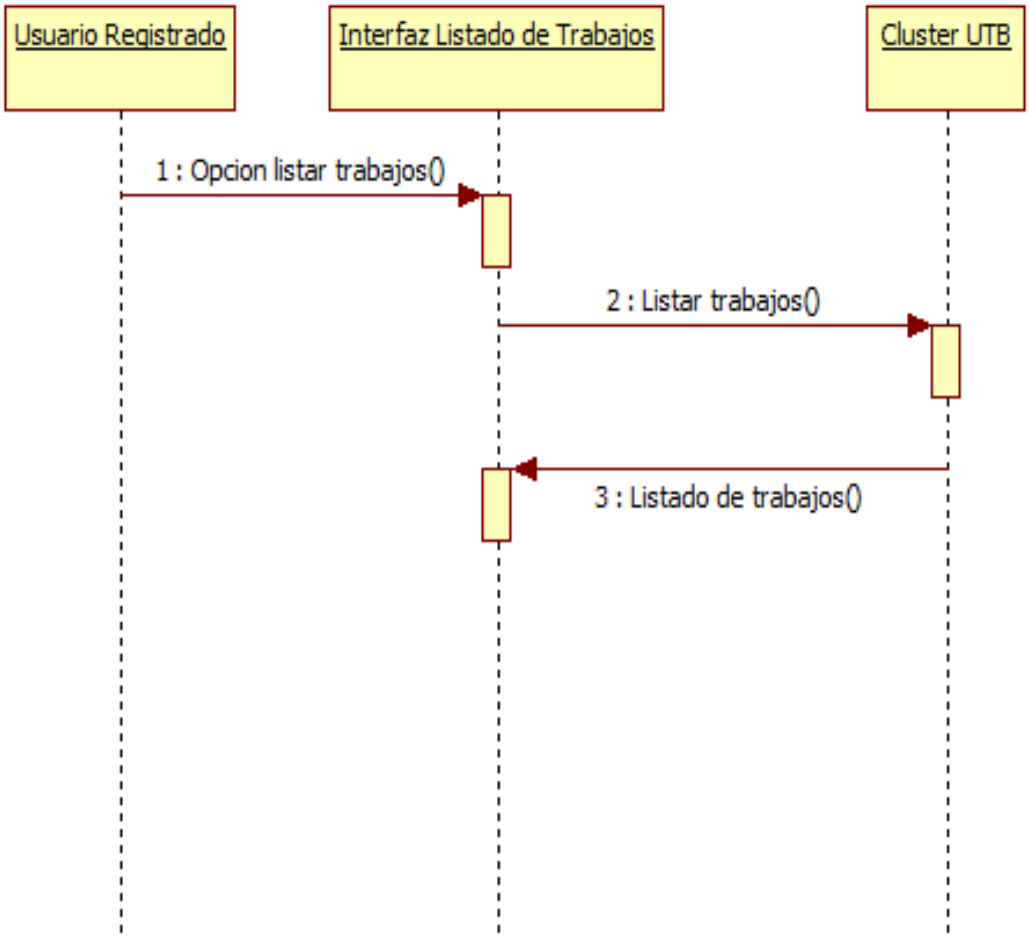
- Editar trabajos



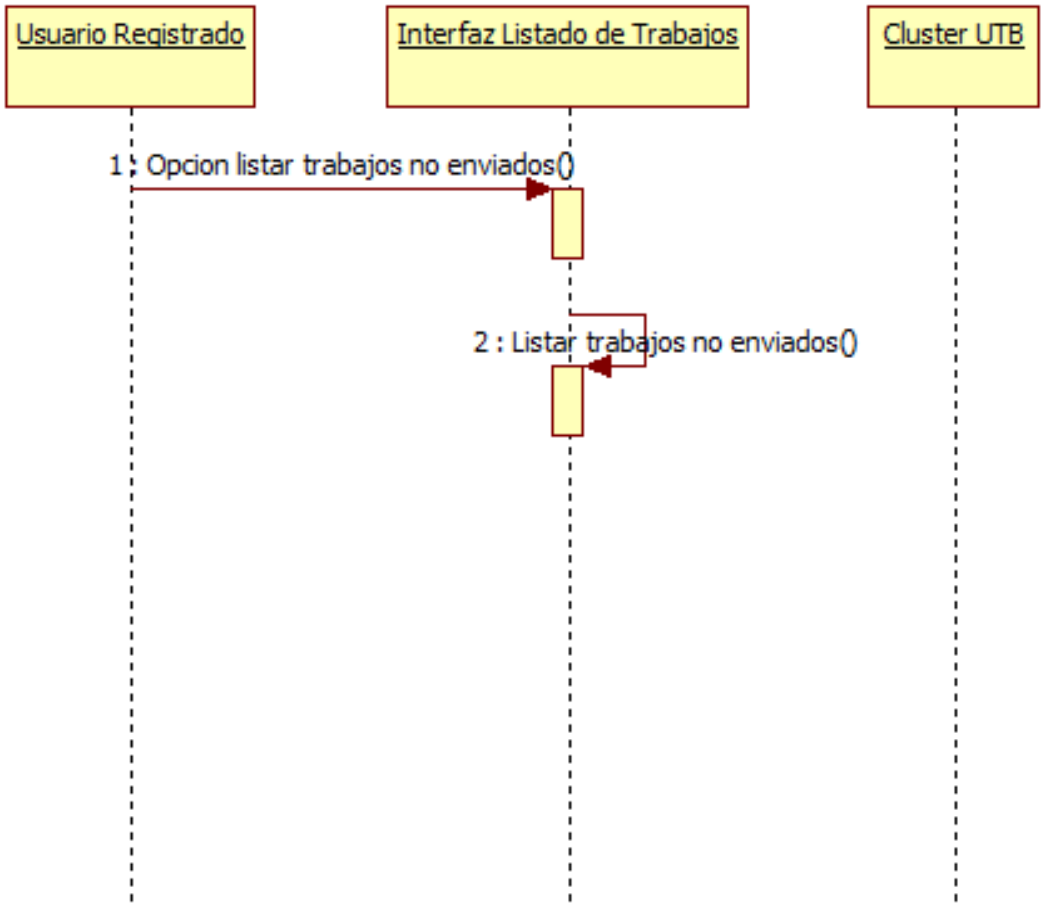
- Enviar trabajo



- Listar trabajos

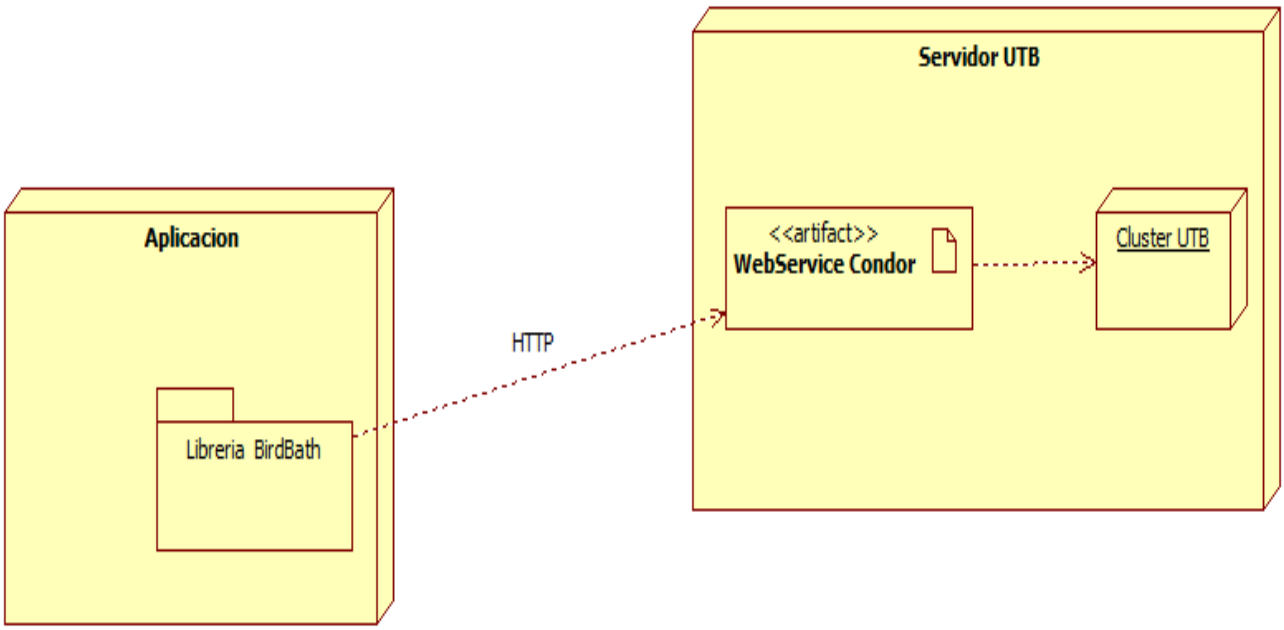


- Listar trabajos no enviados.



6.3.5. Diagrama de despliegue

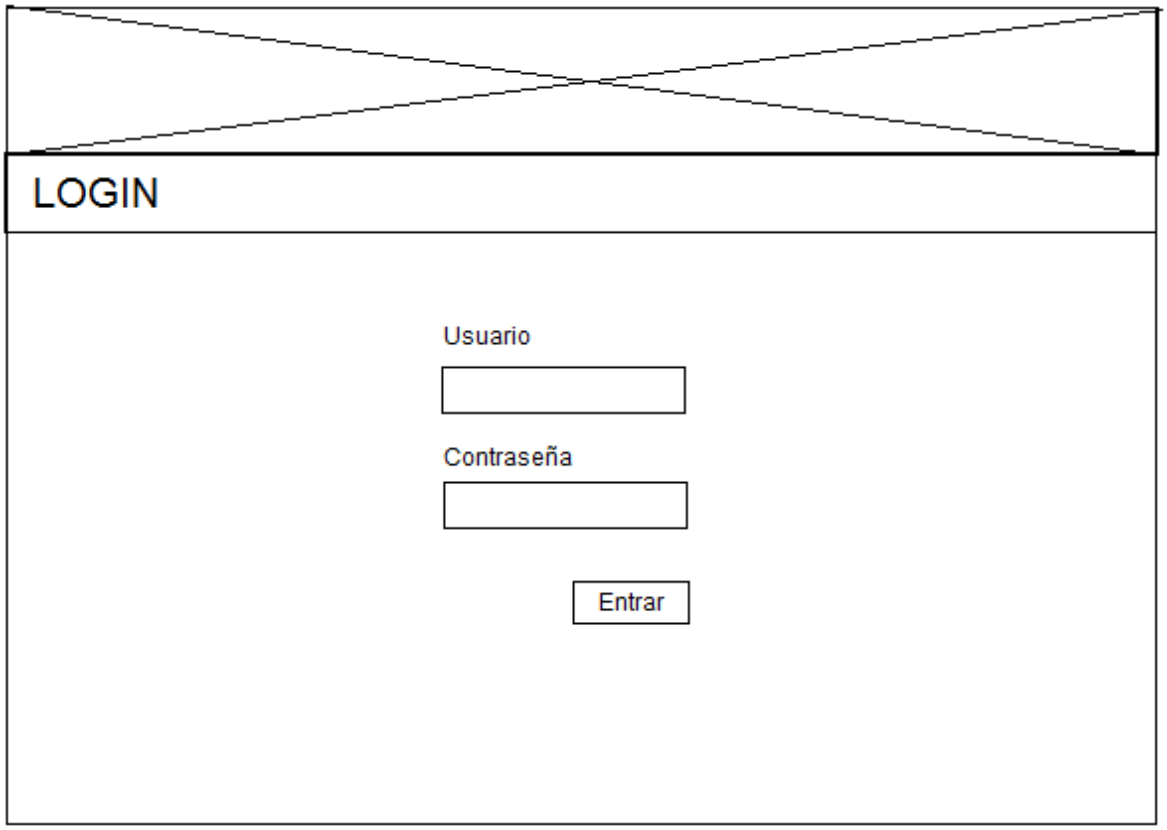
.



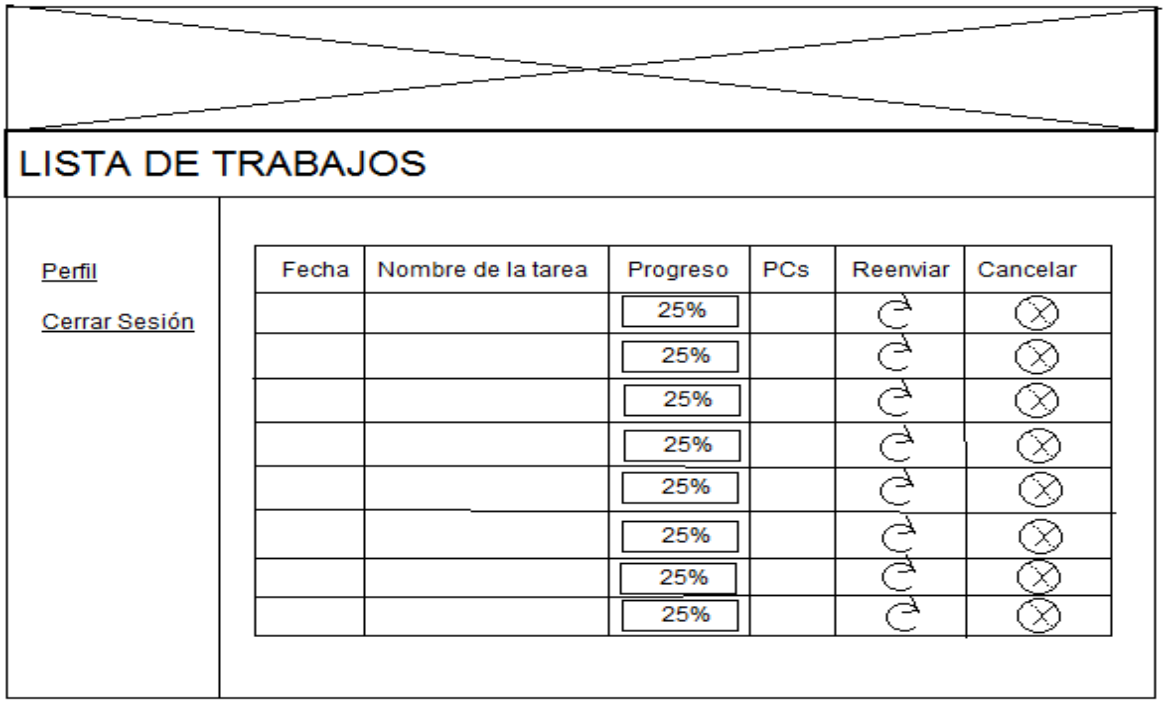
6.3.6. Interfaz de Usuario

El aplicativo podría tener una interfaz como la siguiente:

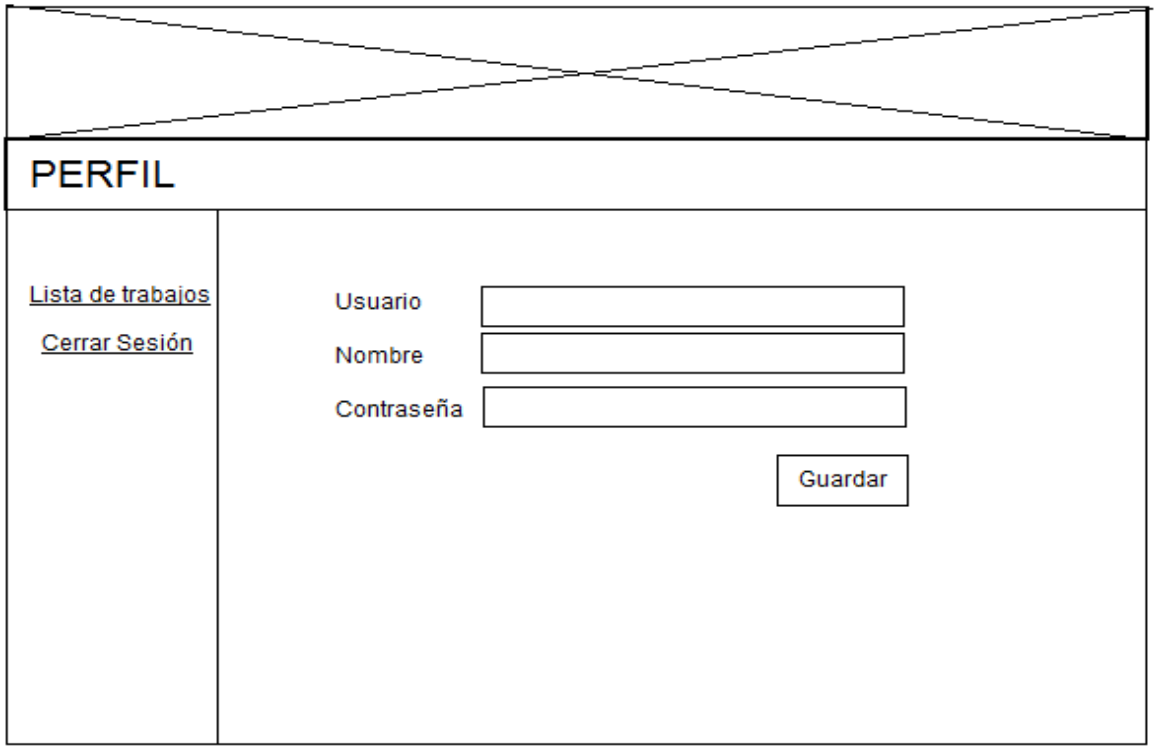
- Pantalla de Login



- Pantalla de Listado de Trabajos



- Edición de Perfil de Usuario



6.4. Cluster de procesamiento de la UTB

El cluster de procesamiento que tiene la Universidad Tecnológica de Bolívar consta de tres maquinas: Alpha, Medusa, Grid y GridUI. Las cuales poseen las características mencionadas a continuación.

Alpha

- Servidor Hewlett-Packard Proliant 785L.
- 32 núcleos Opteron para procesamiento.
- 2500 GHZ de procesamiento.
- 2 TB de almacenamiento.
- 256 GB de memoria.
- Conexión gigabit.
- Sistema operativo Opensuse 12.1.
- Software: MPI, GCC/Fortran.

Medusa

- Laboratorio de computadoras que consta de 10 nodos de 2 GB de memoria cada uno.
- Software: MPI

Grid

- 16 núcleos Intel para procesamiento.
- 3600 GHZ de procesamiento.
- 1 TB de almacenamiento.
- 16 GB de memoria.
- Conexión gigabit.

- Sistema operativo Scientific Linux 6.
- Software: Condor, GCC.

GridUI

- 2 núcleos Intel para procesamiento.
- 1 GHZ de procesamiento.
- 0.1 TB de almacenamiento.
- 4 GB de memoria.
- Conexión gigabit.
- Sistema operativo Scientific Linux 6.
- Software: Condor, GCC.

En el wiki que provee la Universidad Tecnológica de Bolívar se podrá encontrar toda la información referente al clúster además de las características previamente mencionadas así como la instalación y configuración del mismo. (http://hpclab.unitecnologica.edu.co/)

7. CONCLUSIONES

Con el trabajo realizado en el presente proyecto, se han investigado y analizado diversas herramientas con las cuales se puede realizar computación paralela, tanto infraestructuras ya definidas e implementadas, como diversas librerías las cuales soportan el desarrollo en paralelo. Cada una de estas tecnologías pueden ser aplicadas en distintos escenarios, y resolver un problema de diversas maneras, pero se conoce también, que para un escenario especifico, cada una de estas tecnologías posee ventajas y/o desventajas sobre las demás, por lo que se debe realizar un análisis exhaustivo de cada una de estas herramientas enmarcadas en el escenario a resolver, para luego elegir la más factible.

La importancia en la capacidad y velocidad de procesamiento en la resolución de procesos diversos es fundamental en un desarrollo automatizado.
Muchos de estos procesos necesitan tiempos cortos de procesamiento, otros procesos pueden soportar tiempos medianamente largos, pero cuando se obtienen tiempos muy largos de procesamiento, estos pueden ser disminuidos utilizando grid computing, cuya difusión en estos últimos tiempos ha crecido, y se está utilizando en diversos proyectos, tanto de investigación, como en procesos empresariales.

Este crecimiento en entornos de aplicación Grid permitirá el uso de recursos computacionales que se encontraban fuera del alcance de las organizaciones, lo que tendrá un alto impacto de desarrollo propicio para implementaciones científicas a gran escala e integración de servicios entre sectores empresariales.

La implementación de tecnologías Grid permite resolver una gran cantidad de problemas a través de la integración de múltiples organizaciones, lo que significa no solo un ahorro de costos y tiempo sino que garantiza la calidad, gestionando la información de manera que se pueda involucrar aquellas organizaciones que cumplan los requisitos apropiados.

8. RECOMENDACIONES A FUTURO

Como recomendaciones finales, se esperaría que trabajos que necesiten una alta capacidad de procesamiento, sean analizados para estudiar la factibilidad de ser ejecutados mediante Grid Computing para lograr una mayor capacidad y velocidad de procesamiento.

Sería beneficioso para su conocimiento sobre Grid Computing que se conecte a una de las diversas Grid que existen en apoyo a la ciencia, matemática, biología, etc. Ya que de esta forma apreciarían con mayor claridad cómo funciona una Grid y también se podría conocer una de sus muchas aplicaciones que tiene dicha tecnología informática.

REFERENCIAS

[1] The Grid: Blueprint for a new computing infrastructure, Carl Kesselman e Ian Foster, 1998.

[2] ADMINISTRADOR DE PROYECTOS DE GRID COMPUTING QUE HACEN USO DE LA CAPACIDAD DE CÓMPUTO OCIOSA DE LABORATORIOS INFORMÁTICOS, Martín Alberto Ibérico Hidalgo

[3] [FRANK BUSCHMANN,KEVLIN HENNEY, DOUGLAS C. SCHMIDT (2007): Pattern-Oriented Software Architecture A Pattern Language for Distributed Computing. Vol 4. John Wiley & Sons Ltd. ISBN-13: 978-0-470-05902-9

[4] Wikipedia, Defincion de LHC y CERN; http://es.wikipedia.org/wiki/LHC; http://es.wikipedia.org/wiki/Organizaci%C3%B3n_Europea_para_la_Investigaci%C3%B3n_Nuclear

[5] The Grid Idea and its Evolution; Gregor Von Laszewski, Laboratorio nacional de Argonne, USA.

[6] The Evolution of the Grid; David De Rouve, Nicholas R. Jennigs and Nigel Shadbolt, Univesidad de Southampton – UK; Mark A. Baker, Universidad de Porsmooth – UK.

[7] The Grid: Platform for e-Science, e-Business and e-Life; Wolfgan Gentzsch, MCNC.

[8] The organization and management of grid infrastructures; Ian Bird and Bob Jones, CERN; Kerk F. Kee, Universidad de Texas.

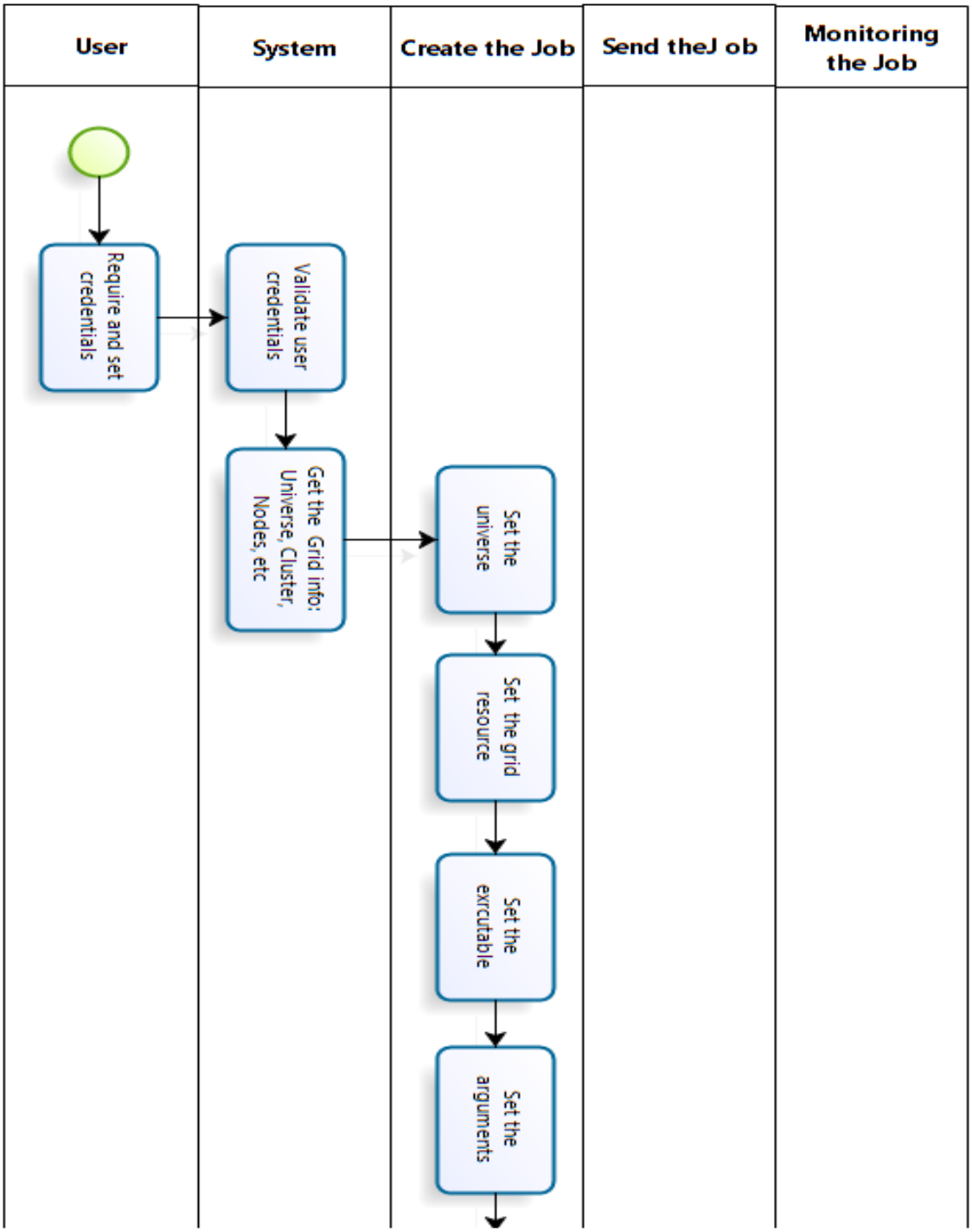
[9] Grid Architecture From a Metascheduling Perspective; Eduardo Huedo, Ruben S. Montenegro, Ignacio M. Llorente, Universidad computacional de Madrid.

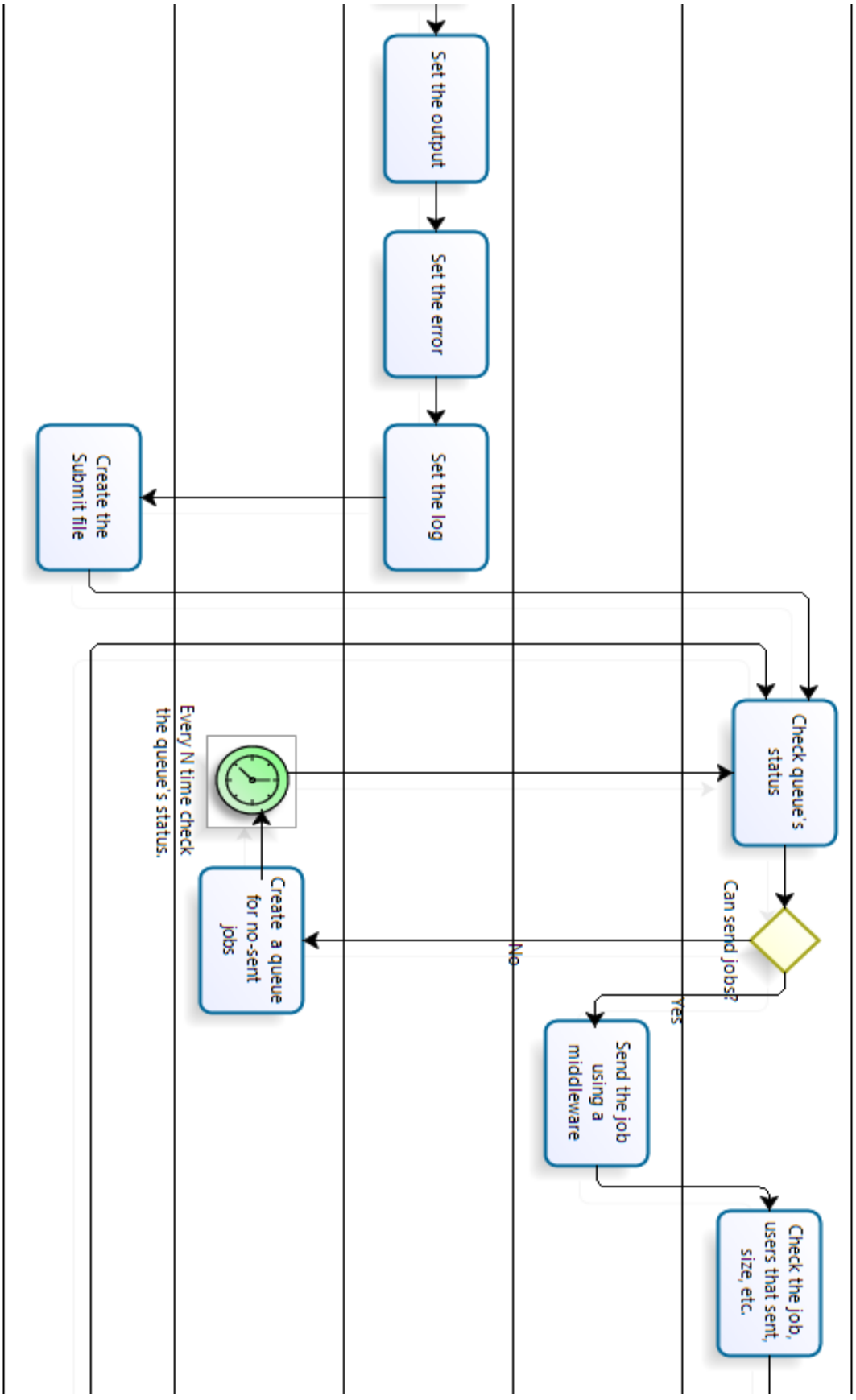
[10] Tesis COMPUTACIÓN DISTRIBUIDA: GRID COMPUTING, Benjamín Domínguez Hernández, Guatemala.

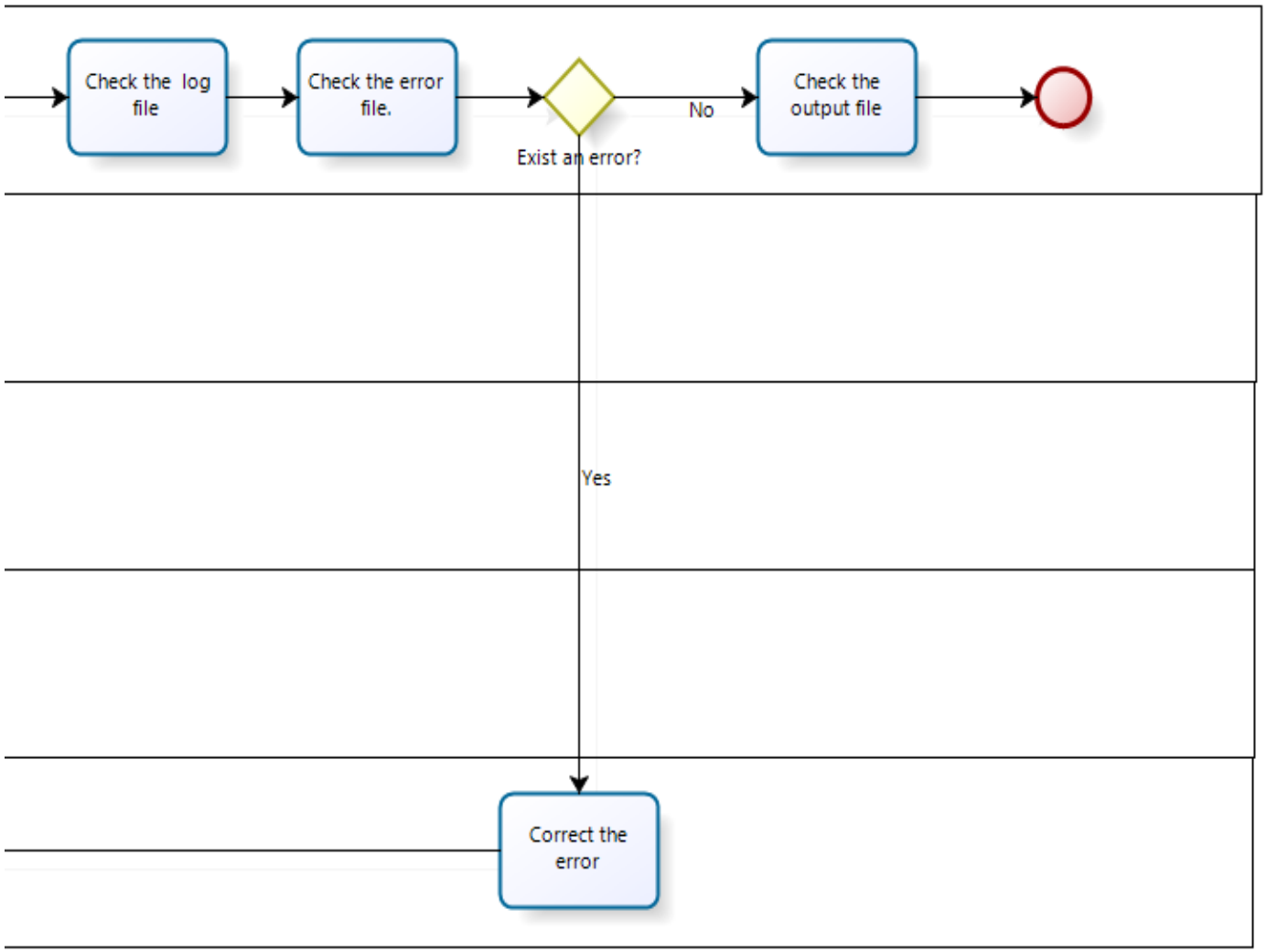
[11] Tesis Modelo de Programación para Infraestructuras Grid Computacionales, José Herrera Sanz, Madrid 2008..

ANEXOS

Anexo A. Diagrama BPM




Set the output
Set the error
Set the log
Create the Submit file
Check queue's status
Can send jobs?
No
Yes
Every N time check the queue's status.
Create a queue for no-sent jobs
Send the job using a middleware
Check the job, users that sent, size, etc.



Anexo B. Casos de uso

C01	**Validar login**	
Resumen	Este caso de uso inicia cuando el sistema necesita validar datos enviados desde un formulario. El sistema recibe los datos, aplica las validaciones necesarias y devuelve un mensaje de error si los datos no pasaron las validaciones o un mensaje de éxito si los datos fueron correctos.	
Prioridad	Alta.	
Frecuencia De Uso	Alta.	
Precondiciones	Ninguna.	
Condición Final De Éxito	Acceso al aplicativo.	
Condición Final De Fracaso	No podrá acceder al aplicativo.	
Actores Directos	Usuario que desea acceder.	
Actores Secundarios	Ninguno.	
Disparador	El sistema necesita validar las credenciales de acceso de un usuario.	
Casos Incluidos	Ninguno.	
Flujo Principal	**Paso**	**Acción**
	1	El usuario digita sus credenciales de acceso.
	2	El sistema valida si el usuario esta registrado en el clúster de la UTB.
	3	Las credenciales pasan las validaciones.
	4	El usuario entra al aplicativo.
Flujo Alterno	**Paso**	**Acción Alterna**
	3.1	Los credenciales de acceso no pasan las validaciones.
	3.2	El sistema muestra un mensaje de error.

C02	**Editar perfil de usuario**	
Resumen	Este caso de uso inicia cuando el usuario hace click en el botón editar perfil. El sistema desplegara un formulario en el cual se le mostrara al usuario los campos que podrá editar. El sistema actualizara los datos del perfil de usuario en el clúster de la UTB.	
Prioridad	Alta.	
Frecuencia De Uso	Alta.	
Precondiciones	El usuario debe estar creado y habilitado en el clúster de la UTB.	
Condición Final De Éxito	Se actualiza un perfil de usuario existente.	
Condición Final De Fracaso	La solicitud para la edición del perfil de usuario es rechazada.	
Actores Directos	Usuario del aplicativo.	
Actores Secundarios	Ninguno.	
Disparador	El usuario pulsa el botón de editar perfil.	
Casos Incluidos	Validar datos.	
Flujo Principal	**Paso**	**Acción**
	1	El usuario hace clic en el botón editar perfil.
	2	El sistema despliega un formulario.
	3	El usuario llena el formulario y lo envía.
	4	El sistema valida los datos ingresados.
	5	El sistema actualiza un perfil de usuario existente
	6	El sistema muestra un mensaje de éxito al usuario.
Flujo Alterno	**Paso**	**Acción Alterna**
	4.1	El sistema rechaza la solicitud para editar los datos básicos del usuario.
	4.2	El sistema muestra un mensaje de error indicando porque no se pudo editar los datos básicos del usuario.

C03	**Crear trabajos**	
Resumen	Este caso de uso inicia cuando el usuario hace clic en el menú Trabajos, submenú Crear. El sistema desplegara una ventana donde el usuario debe especificar las propiedades del trabajo que desea crear.	
Prioridad	Alta.	
Frecuencia De Uso	Alta.	
Precondiciones	El usuario debe estar creado y habilitado en el clúster de la UTB.	
Condición Final De Éxito	Se crea un trabajo exitosamente.	
Condición Final De Fracaso	La solicitud para la creación del trabajo es rechazada.	
Actores Directos	Usuario del aplicativo.	
Actores Secundarios	Ninguno.	
Disparador	El usuario hace clic en el menú Trabajos, submenú Crear.	
Casos Incluidos	Validar datos.	
Flujo Principal	**Paso**	**Acción**
	1	El usuario hace clic en el menú Trabajos, submenú Crear.
	2	El sistema despliega una ventana con un formulario con todas las propiedades del trabajo.
	3	El usuario llena el formulario y lo envía.
	4	El sistema valida los datos ingresados.
	5	El sistema guarda el trabajo creado.
	6	El sistema muestra un mensaje de éxito al usuario.
Flujo Alterno	**Paso**	**Acción Alterna**
	4.1	El sistema rechaza la solicitud para crear el trabajo.
	4.2	El sistema muestra un mensaje de error indicando porque no se pudo crear el trabajo.

C04	**Editar trabajos**	
Resumen	Este caso de uso inicia cuando el usuario hace clic en el menú Trabajos, submenú Editar. El sistema desplegara una ventana con el listado de todos los trabajos creados, el usuario debe seleccionar un trabajo para modificar sus propiedades.	
Prioridad	Alta.	
Frecuencia De Uso	Alta.	
Precondiciones	El usuario debe estar creado y habilitado en el clúster de la UTB.	
Condición Final De Éxito	Se edita un trabajo exitosamente.	
Condición Final De Fracaso	La solicitud para la edición del trabajo es rechazada.	
Actores Directos	Usuario del aplicativo.	
Actores Secundarios	Ninguno.	
Disparador	El usuario hace clic en el menú Trabajos, submenú Editar, selecciona un trabajo.	
Casos Incluidos	Validar datos.	
Flujo Principal	**Paso**	**Acción**
	1	El usuario hace clic en el menú Trabajos, submenú Editar.
	2	El usuario selecciona un trabajo de la lista de trabajos creados.
	3	El sistema despliega una ventana con un formulario con todas las propiedades del trabajo.
	4	El usuario edita el formulario y lo envía.
	5	El sistema valida los datos ingresados.
	6	El sistema guarda el trabajo modificado,
	7	El sistema muestra un mensaje de éxito al usuario.
Flujo Alterno	**Paso**	**Acción Alterna**
	5.1	El sistema rechaza la solicitud para editar el trabajo.
	5.2	El sistema muestra un mensaje de error indicando porque no se pudo editar el trabajo.

C05	**Enviar trabajos**	
Resumen	Este caso de uso inicia cuando el usuario hace clic en el menú Trabajos, submenú Enviar. El sistema desplegara una ventana con el listado de todos los trabajos creados, el usuario debe seleccionar un trabajo para enviarlo a ser procesado.	
Prioridad	Alta.	
Frecuencia De Uso	Alta.	
Precondiciones	El usuario debe estar creado y habilitado en el clúster de la UTB.	
Condición Final De Éxito	Se envía un trabajo exitosamente.	
Condición Final De Fracaso	El trabajo no puede ser enviado para su procesamiento y se almacena en una cola de trabajos no enviados.	
Actores Directos	Usuario del aplicativo.	
Actores Secundarios	Ninguno.	
Disparador	El usuario hace clic en el menú Trabajos, submenú Enviar, selecciona un trabajo.	
Casos Incluidos	Validar datos.	
Flujo Principal	**Paso**	**Acción**
	1	El usuario hace clic en el menú Trabajos, submenú Enviar.
	2	El usuario selecciona un trabajo de la lista de trabajos creados.
	3	El sistema envía el trabajo.
	4	El sistema muestra un mensaje de éxito al usuario.
Flujo Alterno	**Paso**	**Acción Alterna**
	3.1	El sistema no puede enviar el trabajo.
	3.2	El sistema agrega el trabajo a una cola de trabajos no enviados.

C06	**Listar trabajos**	
Resumen	Este caso de uso inicia cuando el usuario hace clic en el menú Trabajos, submenú Listado. El sistema desplegara una el listado de todos los trabajos enviados a procesar así como el estado en que se encuentran.	
Prioridad	Alta.	
Frecuencia De Uso	Alta.	
Precondiciones	El usuario debe estar creado y habilitado en el clúster de la UTB.	
Condición Final De Éxito	Se muestra la lista de trabajos.	
Condición Final De Fracaso	No se puede mostrar la lista de trabajos.	
Actores Directos	Usuario del aplicativo.	
Actores Secundarios	Ninguno.	
Disparador	El usuario hace clic en el menú Trabajos, submenú Listado.	
Casos Incluidos	Validar datos.	
Flujo Principal	**Paso**	**Acción**
	1	El usuario hace clic en el menú Trabajos, submenú Listado.
	2	El sistema despliega el listado de todos los trabajos enviados a procesar así como el estado en que se encuentran.
Flujo Alterno	**Paso**	**Acción Alterna**
	2.1	El sistema no puede mostrar el listado de trabajos.
	2.2	El sistema muestra un mensaje indicando el error.

C07	**Listar trabajos No Enviados**	
Resumen	Este caso de uso inicia cuando el usuario hace clic en el menú Trabajos, submenú Listado de Trabajos en Cola. El sistema desplegara una el listado de todos los trabajos que no pudieron ser enviados al cluster de la UTB.	
Prioridad	Alta.	
Frecuencia De Uso	Alta.	
Precondiciones	El usuario debe estar creado y habilitado en el clúster de la UTB.	
Condición Final De Éxito	Se muestra la lista de trabajos en cola.	
Condición Final De Fracaso	No se puede mostrar la lista de trabajos en cola.	
Actores Directos	Usuario del aplicativo.	
Actores Secundarios	Ninguno.	
Disparador	El usuario hace clic en el menú Trabajos, submenú Listado de Trabajos en Cola.	
Casos Incluidos	Validar datos.	
Flujo Principal	**Paso**	**Acción**
	1	El usuario hace clic en el menú Trabajos, submenú Listado de Trabajos en Cola.
	2	El sistema despliega el listado de todos los trabajos que no pudieron ser enviados a procesar así como el estado en que se encuentran.
Flujo Alterno	**Paso**	**Acción Alterna**
	2.1	El sistema no puede mostrar el listado de trabajos en cola.
	2.2	El sistema muestra un mensaje indicando el error.

Anexo C. Diagramas de clases librería BirdBath