BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

Herramienta de software embebido para la planificación de procesos masivos en tiempo real.

TESIS PROFESIONAL
para obtener el título de
LICENCIADO EN **CIENCIAS DE LA COMPUTACIÓN**

PRESENTA
José Felipe Ortega Pérez

ASESOR:
M.C. Mariano Larios Gómez

Febrero 2019

Dedicatoria:

Mi Tésis la dedico con todo mi amor y cariño a mis padres Felipe Ortega Juárez y en un gesto muy especial a mi adorada madre Ma de Los Ángeles Pérez Zoquiapa, quienes con mucho sacrificio y esfuerzo confiaron en mi, me ayudaron a darme una carrera profesional y bendecido por Dios hoy tengo la oportunidad de agradecerles aunque mi madre ya no está conmigo sino en el cielo, sé que desde allá ella me observa y está contenta por este logro que gracias a su gran valentía, esfuerzo y constancia podré obtener mi Título Profesional.

A mis hermanos y hermanas, Maria, Ana Lilia, Pedro, Guadalupe, Rosario, y Jorge, quienes me apoyaron de manera inconicional, me insitaron a que continuara con mis estudios a que no los abandonara y hoy se refleja en un úmtimo esfuerzo para obtener mi Título Profesional.

A mi amada esposa Ma de los Ángeles López Rámirez, quién con su ayuda incondicional, sus consejos, su insistencia de continuar con mis estudios hasta lograr terminar.

Dedicado a mi adorada madre:
Ma de los Ángeles Pérez Zoquiapa

Agradecimientos:

Agradezco infiniamente a mis padres por impulsar mi desarrollo varios ámbitos, en lo que destacan lo personaly lo profesional a través de oportunidades que me dieron para lograr este trinfo. Asi mismo quiero agradecer principalmente a Dios por dar vida a mis padres y desde luego por darme la vida a mi.

Agrdezco nuevamente a Dios y a la vida misma por darme la dicha de tener a tan ejemplares personas que me ayudaron en el trayecto de mi vida estuadiantil y a su ves en mi vida profesional.

Agradezco a mis maestros por darme la oportunidad de expandir mi conocimientos y mis horizontes a través de sus enseñanzas, consejos y oprtunidades que dieron a mi persona.

Agradezco a mis hermanos, por estar simpre ahí conmigo, apoyándome.

Agradezco a mi amada esposa, por estar siempre ayudándome en todo esté tiempo de regreso a clase, pues ya habían pasado varios años desde que me fui de la universidad dejando truncos los estudios, gracias a mi esposa que me insitó a regresar y terminar.

Agradezco de una forma muy espcial a mi adorada madre Ma de los Ángeles Pérez Zoquiapa, por su inmenso esfuerzo, por su valentía, por sus buenos deseos que tuvo para mi persona, por su lucha incanzable, por su entrega entera para con sus hijos, por todo los sufrimientos que ella tuvo para apoyar a sus hijos entre ellos mi persona, por la lágrimas que mi madre derramo cuando el panorama enconomico se pintó de negro y gris en la casa, con todo mi amor y mi cariño para la persona que lo dio todo, que luchó como una verdadera guerrera para sus hijos, mi madre…

Aternamente agradecido a mi madre:
Ma de los Ángeles Pérez Zoquiapa.

Índice general

Índice de Ilustraciones

Índice de Tablas

NO SE ENCUENTRAN ELEMENTOS DE TABLA DE ILUSTRACIONES.

1 Introducción

1.1 Antecedentes

Este proyecto de tesis es parte del proyecto de investigación en el laboratorio de supercómputo de la BUAP-LNS (Laboratorio nacional del sureste de México), con número 201801076C. Gracias que fue aceptado, el grupo de investigadores y alumnos, se realizaron las pruebas propuesta en los objetivos y metas para obtener los resultados que dieron pie a la obtención de tesis como en [1] y la aquí presente. Además de las publicación de trabajos en un congreso internacionales como [2 y 3].

Institución: Laboratorio Nacional de Supercómputo del Sureste de México LNS
Nombre del Proyecto: 201801076C. JScheduling: Software embebido para la planificación en tiempo real de procesos masivos
Fuente de Financiamiento: LNS-BUAP

En este proyecto se diseñó y desarrollaró un sistema embebido, enfocado a la mejora de tráfico de información entre múltiples dispositivos móviles, múltiples proceso y múltiples tareas. El impacto de esta investigación se enfocó en ambientes de dispositivos móviles distribuidos, como pueden ser drones, teléfonos portables, automóviles o dispositivos móviles re-programables que se encuentran en hogares, oficinas y espacios públicos.
El enfoque del proyecto consiste en el diseño e implementación de una herramienta visual que permite representar la planificación de procesos en tiempo real, ejecutando un número considerable de tareas en dispositivos móviles, mediante esta interfaz se visualiza la localización de los dispositivos móviles dentro de la red. Este proyecto se simula en una arquitectura de cómputo de alto rendimiento (HPC

por sus siglas en inglés), principalmente para lograr visualizar el rendimiento del software embebido para la planificación de tareas en un sistema de cómputo.

La importancia de los sistemas distribuidos es relevante en las ciencias computacionales. Hoy en día se encuentran como parte fundamental en paradigmas complejos, por ejemplo en sistemas ubicuos, sistemas pervasivos, sistemas de tiempo real, robótico, inteligencia artificial, cómputo móvil, entre otros.

Los sistemas de tiempo real y sistemas distribuidos, son fuertemente incluidos y utilizados en la última generación de aeronaves, naves espaciales, control de robots, comunicación de dispositivos móviles inalámbricos (DMI), etc. En los DMI necesitan algoritmos de tiempo real por que cuentan capacidad reducidas de respuesta y la comunicación debe ser inmediata por tener restricciones con respecto al uso de sus recursos limitados [3]. En un sistema distribuido (SD), se tienen nodos como un sistema dinámico y/o un sistema estático, por ende es necesario realizar el análisis de la métrica estándar a utilizar, la cual es la minimización de la suma del peso ponderado en tiempos determinados, basados en un costo de enviar un paquete entre dos nodos. Esta métrica es importante puesto que en los DMI la diferencia de valores se imparte sobre el sistema complementados [4]. Esta métrica propuesta está basada en [5], donde compara diferentes algoritmos basados en teoremas de planificadores de procesos sobre uniprocesadores y multiprocesadores midiendo el tiempo de cómputo requerido para la determinación de un planificador que satisface el orden parcial y las restricciones de los recursos.

En este trabajo de investigación se implemento un algoritmo de planificación en tiempo real [1]. Tal algoritmo fue diseñado e implementado para la obtención de resultados asignando tareas basadas en un mecanismo de consenso entre varios nodos móviles de un sistema distribuido. Tomando en cuenta la calidad de retardo y la restricción del tiempo como una métrica para la evaluación de rendimiento,

permitiendo así, la forma en que los datos del dispositivo móvil puedan ser transferidos y localizados en una red sin pérdida de información.

Una plataforma similar es la que se presenta en [2], la cual busca errores antes de realizar un envió con pruebas automatizadas que se ejecutan en los dispositivos que utilizan los clientes. Las pruebas fueron realizadas en el laboratorio de cómputo de alto rendimiento LNS-CUAP, para realizar pruebas de rendimiento y que permitieron simplificar el desarrollo de la herramienta para cada módulo programado.

En lo académico y en la investigación es necesario llevar al estudiante o joven investigador a la aplicación de metodologías de investigación que se basa en un principio en el estudio bibliográfico que conlleva en el ámbito de los sistemas distribuidos móviles bajo condiciones de tiempo real, con el fin de realizar un análisis y diseño de una red Ad Hoc basada en móviles y algoritmos de protocolo reactivos, dependiendo de la conveniencia de la aplicación. Por tal razón se contempla aplicar algoritmos en una plataforma capaz de ejecutarlos en tiempo real.

La importancia de los sistemas distribuidos es relevante en las ciencias computacionales. Hoy en día se encuentran como parte fundamental en paradigmas complejos, por ejemplo en sistemas ubicuos, sistemas pervasivos, sistemas de tiempo real, en la robótica, inteligencia artificial, cómputo móvil, entre otros. Los sistemas de tiempo real, son incluidos y utilizados en la última generación de aeronaves, naves espaciales, control de robots y comunicación de dispositivos móviles inalámbricos.

En este proyecto se contempla las tareas periódicas, aperiódicas y esporádicas siendo un sistema dinámico y por la inherente espera de una respuesta adecuada en un lapso de tiempo, cumpliendo el deadline en una cuota de tiempo proporcionada por el planificador del sistema. Tal sistema ejecuta un considerable conjunto de dispositivos móviles simulados en un ambiente estrictamente adecuado. Para esto, el SD soportar con un poder de cómputo razonablemente grande en capacidades, de tal forma que se podrá contemplar los resultas de los

algoritmos en el ambiente simulado adecuado. De esta manera se ejecutaran un número masivo de tareas con un número reducido de procesos y de esta forma probar la plataforma propuesta.

La meta a seguir en este trabajo de investigación, es en primera instacia el desarrollar la plataforma embebida que soporte el análisis de un sistema distribuido móvil, aplicando una conexión en red basada en nodos p2p. Una vez obtenida esta plataforma se diseñará e implementará un algoritmo de planificación con el propósito de observar la calidad de retardo.

La principal motivación para realizar este proyecto de investigación de tesis, fue la aportación de una documentación completa y sobre todo la generación de material de investigación y de educación de la programación concurrente y distribuida, esta se justifica aplicando la concurrencia en los sistemas de tiempo real y sobre todo por el poco material que existe en las universidades sobre el tema, en especial en la facultad de ciencias de la computación en la BUAP. Esto nos crea la necesidad de crear más material y documentación para ayudar a mejorar el aprendizaje y entendimiento de la importancia que tiene el paradigma de cómputo concurrente.

1.2 Objetivos

El objetivo principal fue el diseñar e implementar una plataforma embebida en cómputo de alto rendimiento que nos permitio analizar las restricciones en tiempo para llegar al deadline de las tareas en un ambiente distribuido simulado, logrando así, obtener más técnicas y/o métodos para mejorar la gestión y administración de recursos libres y restringidos por el sistema. Así también, se pretende la comunicación en diferentes espacios de recursos compartidos en una configuración distribuida, que permita una topología dinámica a un nodo llamado ***Fenix*** de al supercomputadora LNS-BUAP de forma remota.

Objetivos específicos:

- Diseño y desarrollo de una plataforma embebida con las características de analizar planificadores de tareas preemtives y no-preemtives.
- Conmutación de contexto y extracción en recursos restringidos, precedencias restringidas y tiempo restringido.
- Comunicar a dos o más nodos con multilenguaje y multiplataforma.
- Soportar diferentes algoritmos de planificación y de selección de líder por consenso [6, 7].

1.3 Descripción del problema

Teniendo en cuanta el funcionamiento de los planificadores de tareas en tiempo real viéndolo como una función con respecto al tiempo $\sigma(t)$: $R^{+} \rightarrow N$, se tiene el problema de las restricciones de tiempo, recursos y precedencia de tareas asignadas a proceso y utilización de recursos, es por tal motivo que se propone una plataforma capaz de analizar resultados en tiempo real para observar la falta o asignación de los tiempos limitados (deadline). Esto nos conlleva a un mejor proponer mejoras en algoritmos para obtención de resultados destacados.

La metodología a seguir es la creación de una herramienta embebida en una máquina virtual que cumpla con los requisitos adecuados para la medición del planificador de procesos.

Se diseñará utilizando librerías de la máquina virtual, teniendo en cuenta que no se desea saturara el diseño distribuido y que este fuera concebido al momento de procesos sincronizados [9, 10]. Apoyados con la teoría de selección de un agente de software para analizar las restricciones del sistema [3, 4, 5]

Este trabajo de investigación, se desarrolló bajo el paradigma de cómputo distribuido aplicando una conexión en red basada en nodos p2p. Una vez obtenido este resultado se diseñó y desarrolló los algoritmos de planificación con el propósito

de observar la calidad de retardo. Por último, se diseñó e implementó un algoritmo de ruteo para la distribución de cargas.

1.4 Herramientas embebidas

Las herramientas utilizadas y metodología que se siguió están dirigidas a sistemas embebidos o máquinas virtuales que cumpla con los requisitos adecuados para la medición del planificador de procesos y obtención de los resultados adecuados.
Se diseñó librerías para la máquina virtual, teniendo en cuenta que no se deseó saturar el diseño distribuido y que este fuera concebido al momento de procesos sincronizados [13, 14].
Por otro lado, se aplicó conceptos de lenguajes de programación para utilizar el lenguaje de programación orientado a objetos como Java con métodos nativos e implementan por separado en archivos .c o .cpp y .java por los cuales se generará una biblioteca con enlace dinámico. Utilizando Windows nos genera un archivo .dll el cual enlazamos con nuestra interfaz en java y en Linux genera un .so, como se realizó en [15].

1.5 Utilización de la supercomputadora en el LNS

Se utilizó la siguiente infraestructura con las características específica para poder implementar nuestra investigación y obtener los resultados deseados:

Especificaciones de la supercomputadora
La supercomputadora Cuetlaxcoapan del LNS está compuesta por un clúster estándar de cálculo con procesadores Intel Xeon y un clúster con procesadores Intel Xeon Phi Knights Landing.
Características del clúster Intel Xeon:
228 nodos de cálculo Thin (5472 núcleos totales)
Cada nodo contiene:

- 2 procesadores Intel Xeon E5-2680 v3 (Haswell) a 2.5 GHz

- 12 núcleos por procesador / 24 núcleos totales
- 128 GB de memoria RAM DDR4 a 2133 MHz
- 2 interfaces de red Ethernet Gigabit
- 1 interface de red InfiniBand FDR a 56 Gbps

2 nodos de cálculo con GPU (48 núcleos CPU y 11520 núcleos CUDA totales)

Cada nodo contiene:

- 2 procesadores Intel Xeon E5-2680 v3 (Haswell) a 2.5 GHz por nodo
- 12 núcleos por procesador / 24 núcleos totales
- 128 GB de memoria RAM DDR4 a 2133 MHz
- 2 interfaces de red Ethernet Gigabit
- 1 interface de red InfiniBand FDR a 56 Gbps
- 2 tarjetas GPU NVIDIA Tesla K40
    - 2880 núcleos CUDA por GPU / 5769 núcleos CUDA totales
    - 12 GB de memoria RAM DDR5 por GPU

Almacenamiento:

Se dispone de 1.2 PB de espacio total para almacenamiento en disco

Cluster LUSTRE:

Compuesto por:

- 6 servidores OSS
- 2 servidores MDS
- 1 servidor de almacenamiento MDT de 40 TB
- 3 servidores de almacenamiento OST de 320 TB cada uno

1.6 Redes de Comunicaciones

Se dispone de:

- Una red Ethernet Gigabit para gestión del hardware de la supercomputadora y el aprovisionamiento de software a los nodos.

Una red Infiniband FDR (56 Gpbs) para comunicación rápida entre nodos de cálculo [16].

Se obtuvo una programación que ayudó a medir el tiempo de respuesta de un planificador de proceso y la obtención del tiempo de retardo, esto aplicado a un ambiente distribuido. Dando como consecuencia el poder realizar un prototipo y en una plataforma distribuida.

Esta investigación esta centrada la planificación de procesos de un sistema de software o algoritmos distribuidos, paralelos y de tiempo real.
Se obtuvó una plataforma que ayuda a medir el tiempo de respuesta de un planificador de proceso y la obtención del tiempo de retardo, esto aplicado a un ambiente distribuido móvil. Dando como consecuencia el poder realizar un prototipo y patentar la plataforma y herramientas que se desarrollen.

Plataforma única para la gestión de solicitudes y planificación de procesos de un sistema de software o algoritmos distribuidos, paralelos y de tiempo real. El personal de la BUAP puede utilizar esta herramienta para visualizar los resultados de proyectos que impliquen las áreas mencionadas.
Esta herramienta visual puede planificar procesos que nos permita visualizar los resultados en tiempo real para la asignación de tareas basándose con diversos nodos de un sistema distribuido móvil, y en un ambiente simulado. Los desarrolladores tendrán un mejor control de los dispositivos móviles virtuales. El sistema sirve como referente para el proyecto de LNS con número de SOLICITUD: 201701071C. Gracias es estos puntos se obtendrá un documentación completa y confiable sobre una plataforma embebida cómputo de alto rendimiento.

2 Marco Teórico.

2.1 Estado actual de aplicacion

En este estado del arte nos enfocaremos en los temas de ambientes distribuidos simulados y sistemas embebidos. En [18] se discute la aplicación de los resultados para el problema de sistemas multiagentes en tiempo real, aquí se proponen dos algoritmos basados en la matriz Laplaciana del grafo G de redes que logra el consenso en un tiempo finito usando funciones Lyapunov, de esta manera se propone un mapa distribuido especial para la clase de grafos no dirigidos, y da pie a la propuesta de este proyecto de generar una interfaz con el funcionamiento de grafos no dirigidos, además de proponer un análisis exhaustivo y el diseño de estrategias cooperativas para el consenso, otra contribución es la introducción a las condiciones necesarias y suficiente para dos algoritmos distribuidos discontinuos que logran el consenso mínimo y máximo en tiempo finito y un consenso asintóticamente. Como se propone en este trabajo de investigación de tesis, se plantea el uso de redes Ad Hoc, por tal razón se contemplan estos trabajos antes mencionados, por validar sus resultados en redes con topologías de intercomunicación y cambios dinámicos. Otro trabajo con propuestas novedosas sobre los algoritmos de consenso se encuentran en [19, 20, 21, 22 y 23], en estos artículo se re-direccionó el problema de consenso para sistemas distribuidos multiagentes no lineales y se aplica un controlador a grafos dirigidos y no dirigidos, además este control permite que se pueda trabajar en topologías fijas y cambiables. La aplicación de consensos y acuerdos en multiagentes en ambientes distribuidos y diseño de observadores lo podemos encontrar en [19], este trabajo se basa en las reglas de vecindarios L para la coordinación de multiagentes. En la búsqueda de un líder activo, se describe la dinámica del agente a seguir con entradas interactivas de control.

2.2 Ambientes distribuidos

La sincronización del proceso ocurre cuando el progreso de uno o varios procesos depende del comportamiento de otros procesos. Dos tipos de interacción de proceso requieren sincronización: competencia y cooperación.

De manera más general, la sincronización es el conjunto de reglas y mecanismos que permite la especificación e implementación de propiedades de secuenciación en los enunciados emitidos por los procesos para que todas las ejecuciones de un programa de multiprocesamiento sean correctas.

Un ejemplo de las aplicaciones de los sistemas distribuidos (SD) lo podemos encontrar en [20], este trabajo se enfoca al problema de determinar una ruta óptima; a través de un algoritmo de descubrimiento de rutas. Se aplica el AEC a un ambiente SD empleando una matriz de adyacencia, después se utiliza la distancia numérica para aplicar el consenso para el caso de estudio de topología móvil. Por ejemplo en [8], podemos observar la aplicación del control difuso basándose en la teoría de Takagi-Sugeno, En este experimento fueron basados en plataformas comerciales, aplicando la metodología de Takagi-Sugeno para el control con lógica difusa. Tanto en cuadricóptero y helicóptero se establece tres puntos de regulación.

2.3 Aplicaciones contemporaneas

Estas condiciones son importantes para la obtención de las cuantificaciones deseadas, aunque se limitan a solo tres conjuntos de ellas. Otro trabajo de investigación donde se destaca el análisis y diseño de los sistemas de control en redes NCS, se encuentra en [21], aquí se experimenta un control LQR (Linear-Quadratic Regulator) aplicado a los retardos en un sistema de tiempo real que se encuentra en un NCS. A comparación del trabajo en [21], en [22] se presenta un modelado usando Euler-Lagrange para el helicóptero en 2DoF. Basados en este modelo se aplica un control FF-LQR para controlar un helicóptero regulando al eje

en ángulo pitch con FF. El control FF+LQR+I usa un integrador en el lazo retroalimentado para reducir el error en estado estacionario. Usando la FF y la velocidad integral proporcional (PIV) se regula el ángulo pitch y sólo con la PIV se controla el ángulo de yaw.

El control LQR ajusta de manera efectiva el conjunto de frecuencias sin que el cálculo de la ley de control y la reconfiguración misma impacte en la transición. Es de resaltar que esta última reconfiguración disminuye el índice de desempeño provocada por una utilización de la red fuera de su ancho de banda o debido a bajo muestreo de la transmisión fuera de la región de planificación en los ángulos y velocidades en pitch y yaw, es decir, respectivamente.

Otro trabajo de investigación es presentado en [14], donde se presenta una propuesta para el control del helicóptero. El objetivo es mantener el ángulo deseado de pitch y yaw, a través de dos hélices con dos motores como actuadores, además de otros casos de uso como la simulación de un sistema de levitación magnética, con un electro magneto como actuador, y dos sensores que miden la posición de una bola de acero y la corriente del electro magneto.

Para mostrar la aplicación del codiseño de un NCS se presentan la pérdida de paquetes en una comunicación remota entre computadores o peers, basada en el protocolo UDP.

El control difuso diseñado es aplicado a ambos casos de estudio, utilizando el modelo difuso diseñado y el modelo de las imperfecciones. Se diseñan las matrices de retroalimentación para cada sub modelo discreto a través de un diseño LQR [21, 22].

Un servidor proxy es una solución software, implementada en la capa de aplicación de TCP/IP, el cual intercepta los mensajes protocolarios (ej. HTTP) para realizar la solicitud en representación de los usuarios de la red. Generalmente se encuentran ubicados en la frontera entre una red local y la red del ISP (Internet Servicie Provider) [15].

En otras palabras, un proxy funge como servidor intermediario (o procurador por su traducción al español) para el control del tráfico generado entre dos o más dispositivos en internet, simulando un sustituto de ruteo.

Los servidores proxy se pueden encontrar en redes corporativas, instaladas con un diseño especializado para los dispositivos de una intranet. Y comúnmente utilizados por los ISP como parte se los servicios en línea que proveen a sus consumidores [16].

2.4 Funciones y métodos de comunicación remota

Las funciones claves que un servidor proxy comúnmente provee son:

1. Cortafuegos y soporte para filtrado de tráfico.
2. Conexión compartida de redes.
3. Cache de datos.

Un proxy, como intermediario entre dos dispositivos, permite la solicitud de recursos, por ejemplo, a un servidor web, sin que este sepa a quién va dirigida la información realmente.

A modo de ejemplo, un proxy recibe la consulta de un ‘Host A’ y realiza nuevamente la consulta hacia ‘Servidor B’, bajo un mismo protocolo de comunicación previamente definido, para posteriormente recibir la respuesta y redirigirla de vuelta al ‘Host A’.

Su aplicación más conocida, en la totalidad de los casos, es la ocultación de los dispositivos que realizan las consultas (ej. Host A), comúnmente utilizados en prácticas ilegales como fraudes, suplantación de identidad, spam, etc. Aunque su principal objetivo es la interconexión dedicada entre muchos ordenadores de una misma red a través de internet, como es en el caso de las VPN (Virtual Private Network), que en los últimos años ha tomado popularidad por su comunicación fuertemente cifrada a través de los ISP.

Una VPN es capaz de cifrar el tráfico que pasa a través de él, que a diferencia de un proxy, el tráfico quedaría vulnerable, a menos que se defina un protocolo de cifrado a nivel de capa de aplicación.

Las principales ventajas son la reducción de tiempo de configuración (frente a una VPN), así como el filtrado de paquetes, la comunicación directa, el anonimato y la evitación de restricciones geográficas. Las desventajas son pocas aunque considerables como la implementación ante muchas peticiones puede comprometer su estabilidad y sobrecargarle, el almacenamiento de datos en cache de los ordenadores, que por un lado proporciona un acceso más rápido a datos redundantes, pero logrando comprometer la confidencialidad de los datos trasmitidos, los proxys desconocidos, el cifrado puede ser vulnerado a menos que se utilice una capa SSL (Secure Socket Layer) y la incoherencia o desactualización, si se utiliza almacenamiento de cache, los datos almacenados en el proxy pueden diferir de los datos actualizados en el servidor destino.

2.5 Tipos de proxy

Destacan dos descripciones base que definen los objetivos con el que se desee implementar un servidor proxy.

- Proxys externos

Los más comúnmente usados para proporcionar un servicio a todo internauta desde el que puede realizar una conexión.

2.5.1 Proxys Internos

Estos se instalan con el objetivo de controlar el tráfico y evitar que la información privada de un ordenador o una red interna sean accedidas fácilmente, evitando solicitudes no autorizadas, optimizando los recursos y reduciendo la carga de trabajo de los servidores.

Y que, para fines de este proyecto, este es implementado para el acceso a la red interna del OPA.

2.5.2 Aplicaciones Proxy

De estos se pueden definir con exactitud las algunas aplicaciones proxy.

La caché conserva un contenido solicitado previamente por un usuario para acelerar la respuesta de futuras peticiones de la misma información.

En web su funcionamiento se basa en el uso del protocolo HTTP y HTTPs, con la posible utilidad de proporcionar una cache compartida para las páginas web, actuando como proxy-caché permitiendo liberar la carga para consultas coincidentes de otros usuarios.

Trasparente combina un servidor proxy con cortafuegos de manera que las conexiones son desviadas hacia el proxy sin la necesidad de una con-figuración por parte del cliente y que este último desconozca la existencia del proxy.

NAT utilizado para ser intermediario en una red, las traducciones de red, también conocidas como enmascaramiento de IPs, es una técnica mediante la cual las direcciones IP de origen y destino de los paquetes son reescritas.

Esta aplicación proporciona un cierto nivel de seguridad, puesto que no existe una conexión directa de la red local con el exterior, evitando ataques directos a los equipos internos.

Las peticiones de clientes indistintamente estén o no conectados a su red, más comúnmente utilizado como pasarela de envió masivo de spam. Debido a su uso como almacenamiento y re-direccionamiento de servicios DNS o navegación Web mediante el cacheo, lo vuelve una opción de alta velocidad para el usuario común, aunque lamentablemente su configuración “abierta” para internet lo convierte en una herramienta para actividades ilícitas.

Coss-Domain. Utilizado en tecnologías asíncronas, que poseen restricciones para establecer una comunicación entre elementos localizados en distintos dominios. Un buen ejemplo es AJAX, que por seguridad solo permite acceder a los diversos dominios al implementarse un Cross-Domain en el dominio origen, quien recibe las peticiones para renviar a otros dominios externos.

Inverso. Se sitúa en el dominio de uno o más servidores. Todo el tráfico de internet y con destino a alguno de esos servidores pasa necesaria-mente a través de este esté proxy, sin la intervención del usuario, volviendo su uso trivial en la red.
En este proyecto se trabaja mediante la generación de acceso proxy del tipo in-terno e implementado por un middleware (Globus toolkit) que proporciona una capa de cifrado SSL, parecido una aplicación proxy inversa, para establecer la comunicación con los componentes de cómputo y aplicaciones específicos finales dentro de una red no visibles desde el exterior.

2.6 Tunneling

Método por el cual se establece una comunicación de punto a punto, creando un "túnel" virtual operando bajo IP, permitiendo encapsular todos los paquetes trasportados a través de él con datagramas TCP o UDP, entre los dispositivos que se encuentren en ambos extremos del túnel y pertenezcan a sub redes diferentes.
La comunicación a través del túnel es cifrado mediante protocolos criptográficos, pudiendo ser SSL (Secure Sockets Layer) o TLS (Transport Layer Security). El cifrado proporcionando una capa extra de seguridad por la red en el que se implemente, siendo comúnmente internet.
El uso de túneles, como los del tipo ssh (secure shell), permiten mitigar ataques de red como son:

1. Sniffers: Ataques donde monitorean el tráfico de una red (Wi-Fi común-mente) en busca de datagramas no cifrados y que puedan proporcionar alguna información valiosa.
2. Man in the Middle: donde algún integrante de una red local, el atacante, tiene acceso total a nuestro trafico tanto de entrada como de salida, con la capacidad de leer y modificar a voluntad dicho tráfico que se genere o se reciba.

3. ARP Spoofing: Donde el atacante envenena el cache ARP(Address Resolution Protocol) de la víctima para hacerle creer que el atacante es el router o puerta de enlace, y así recibir la totalidad del tráfico generado pasando por el ordenador del atacante.

Es una práctica común en las organizaciones, el establecer una comunicación con un servidor ssh fuera de una red local donde la seguridad está comprometida (como un café internet), para poder acceder a un servidor remoto y proteger los datos.
Para establecer un canal con la totalidad del tráfico entrante y saliente completamente cifrado es necesario establecer un túnel.

Existen cuatro entidades trabajando en un túnel SSH, que son los siguientes:

1) Aplicación (ej. Navegador Web, lado del cliente)
2) Cliente SSH (Lado del cliente)
3) Servidor SSH (Lado del servidor)
4) Servidor Web (Lado del Servidor)

Por lo tanto, lo que se logra es tener una comunicación segura, mitigando los ataques antes mencionados, y evitar la intercepción del tráfico dentro de una red, garantizando la integridad y confidencialidad de la información trasmitida mediante un cliente SSH, quien toma el tráfico de la aplicación, la cifra y envía a internet mediante el túnel a un servidor SSH, que se encuentra en una ubicación segura y no susceptible a posibles ataques, teniendo en cuenta que en el último tramo de la comunicación (ej. del servidor SSH al Servidor web) se trasmitirá en texto plano y por lo tanto los datos podrían ser interceptados y modificados. Aunque la probabilidad de tener problemas en estos últimos tramos es muy baja.

2.7 Port Forwarding

Es una capacidad del Secure Shell (SSH), el port forwarding (reenvío de puertos) permite que los datos TCP / IP no seguros se trasmitan por el túnel a través de redes públicas y privadas.
Su configuración se realiza mediante el establecimiento de una sesión SSH utilizando el siguiente comando:

Donde la opción **L** indica que esa será un re direccionamiento de puerto local.
A modo de ejemplo se establece un túnel SSH para consultar la página de google de manera segura de la siguiente manera:
El cliente SSH se conectará al servidor SSH ahora mediante el puerto 9001 (siendo por defecto el puerto 22) por el cual "escuchar" las solicitudes. Por último, el servidor SSH crea la conexión con 'google.com' mediante el puerto 80 (HTTP). De esta manera el Servidor SSH actuara como gateway (puerta predeterminada), mientras acepta las peticiones de la maquina cliente SSH, buscando la información y enviándola de vuelta.
Cabe mencionar que el canal entre el cliente y el servidor será cifrado mientras la sesión SSH se mantenga activa [23].
En este proyecto se implementa el tunneling ssh y con redireccionamiento de puertos a través del servidor proxy, para alcanzar, "visualizar", las direcciones IP locales aisladas dentro de la red interna del OPA, permitiendo una comunicación a los recursos que ahí dentro se alojan de forma trasparente.

2.8 Seguridad

Los protocolos de cifrado se usan ampliamente para el trasporte de datos seguros a nivel de aplicación. Y en cuanto a materia de seguridad en redes se habla, es necesario cumplir con tres servicios fundaméntalas para establecer una comunicación fidedigna (Autenticación, Autorización y Cifrado).
Por lo que es importante conocer la evolución y mejora que han tenido los principales métodos de comunicación cifrada desarrollados hasta el momento.

2.8.1 Llave Única

El cifrado mediante de llave única es la versión más básica de estas, ya que es útil, pero con limitaciones en la mayoría de los casos para la trasferencia de datos protegidos.

Debido a la poca sofisticación, donde se utiliza una única llave para cifrar y descifrar en ambos extremos, su uso se ha vuelto poco práctico en aplicaciones como el comercio electrónico. Siendo el método de criptografía de llaves públicas la siguiente solución a la seguridad para toda red abierta.

2.8.2 Llave Pública

En este método es requerida la creación de dos llaves únicas en la comunicación, denominadas; llave pública y llave privada.

La llave pública está a disposición del público, a diferencia de la llave privada que se conserva en secreto. Las dos llaves funcionan en conjunto, donde cualquier dato o información que una de las llaves cierre (cifre), solo podrá abrirse con la otra [30].

Certificados Digitales (SSL/TSL)

SSL (Secure Socket Layer) es un protocolo de propósito general para establecer comunicaciones seguras, propuesto en 1994 por Netscape Communications Corporation junto con su primera versión del Navegador. Y fue hasta su tercera versión, conocida como SSLv3.0 que alcanzo su madurez, superando los problemas de seguridad y limitaciones de sus predecesores.

Actualmente utilizado en el comercio electrónico, pero no es ese su único uso, sirviendo para cualquier comunicación a través de internet. SSL está incorporada en la mayoría de los navegadores web, y hoy día constituye la solución de seguridad implantada en la mayoría de los servidores organizacionales.

SSL está basado en la aplicación conjunta de criptografía simétrica (de llave secreta), criptografía asimétrica (de llave pública), certificados digitales y firmas digitales para conseguir un canal o medio seguro de comunicación [24].

En este método, a diferencia de los dos primeros, es agregada una tercera entidad como mediador de confianza de la comunicación por SSL. Un certificado digital agrega el endoso del tercer miembro, que garantiza la integridad y existencia de la organización que envía y/o recibe los datos. Este tercero es conocido como una autoridad certificadora (CA), esta avala que la empresa y/o sujeto realmente existan. Algunos datos que este documento digital puede contener son:

- Nombre del sujeto
- Llave pública
- Número de serie
- Fecha de expiración
- Firma de la CA
- Ubicación geográfica
- Organización
- De más información relevante, sea el caso de uso

Los certificados pueden ser expedidos para tres tipos de “miembros”:

1) Persona. Expedido explícitamente para el uso de una persona moral, individuo o usuario que utilice dicho certificado para acceder a dispositivos de cómputo y/o servicios en la red.
2) Host. Especialmente diseñado para los dispositivos de cómputo autónomos que requieren ser identificados cada que consuman/ofrezcan algún servicio.
3) Servicio. En este se encuentra la información del servicio específico que se consume, y nombre de dominio del servidor a utilizar, perfecto para implementación de cierto software de diseño.

Existe una jerarquía de autoridades certificadoras, comenzando por la autoridad raíz de certificación, la cual es la Internet Policy Registration Authority (IPRA), esta, firma certificados utilizando la clave raíz, por lo que solo firma certificados para autoridades de creación de políticas (Policy Creation Authorities), que estas a su vez certifican a las organizaciones que implementan estas políticas, las CAs, para expedir certificados digitales a usuarios y/o empresas de alguna organización en específico.

Algunas de las IPRA para la certificación de AC con políticas certificadoras para organizaciones Grid son:

- TAGMPA. La The Americas Grid Policy Management Authority (Autoridad de Gestión de Políticas de Grid de las Américas), es una federación de proveedores de autenticación y partes dependientes, responsables en las redes en el norte, centro y sur América.
- EUgridpma. Es la organización internacional que coordina la red de confianza para la Infraestructura electrónica para la investigación en Europa, Oriente Medio y África.
- APGrid PMA. Soporta las comunidades GRID en Asia Pacífico para implementar un dominio de confianza común entre organizaciones y coordinar la infraestructura de clave pública para su uso con la autenticación Grid.

La CA es la responsable de establecer las políticas necesarias para certificar a las partes, los certificados raíz estarán disponibles públicamente para su uso a la hora de validar un certificado, siendo almacenados por las CA en repositorios o servidores.

La CA firma el certificado mediante el cifrado de la llave pública del sujeto o con un valor hash de la clave pública, utilizando su propia clave privada.

2.8.3 Cifrado

Los principales métodos utilizados conllevan una mejora continua en los fundamentos de cifrado, autenticación y autorización para los protocolos de comunicación por la red.
Un protocolo criptográfico debe incorporar los siguientes aspectos:

- Autenticación de entidades y mensajes
- Integridad de los datos
- Métodos de no repudio
- Confidencialidad

Cuando se corre sobre una sesión SSL, el servidor y el cliente pueden autenticarse el uno al otro, y negociar un algoritmo de comunicación y tipo de cifrado de llaves, antes de que la aplicación tramita o reciba los primeros bytes de datos [25]
SSL es muy flexible en cuanto a la selección de algoritmos de cifrado, funciones de autenticación y métodos de comprobación que se deseen.
En su implementación se pueden utilizar algoritmos de cifrado como:

- DES (Data Encryption Standard)
- IDEA (International Data Encryption Algorithm)
    - Triple DES
    - RC2 y RC4
    - Fortezza
    - RSA (Rivest, Shamir, Adelman)

Para la comprobación de mensajes se puede implementar:

- MD5 (Message Digest Algorithm 5)

- SHA-1 (Secure Hash Algorithm)

A esto se le llama Chiper Suite (Suite de cifrado) y la selección de uno u otro para su implementación depende del nivel de seguridad que se busque tener.
Para establecer una idea clara de que es una arquitectura de cómputo Grid, es importante conocer los conceptos que definen a los sistemas distribuidos, así también el establecer los puntos que diferencian a la Grid con las plataformas homónimas comercial-mente más utilizadas hoy día. Que, aunque sus bases pueden ser similares los niéveles aplicativos de una a otra difieren para el sector que fueron desarrolladas.

2.9 Definición de red

“Sistemas cuyos componentes hardware y software, que están en computadoras conectadas en red, se comunican y coordinan sus acciones mediante el paso de mensajes, para el logro de un objetivo. Se establece la comunicación mediante un protocolo preestablecido.”

2.9.1 Características

- Concurrencia. Un sistema Distribuido debe permitir que los recursos disponibles en la red puedan ser utilizados simultáneamente por los usuarios y/o agentes que interactúen en la red.
- Carencia de reloj global. La coordinación para la transferencia de mensajes entre los componentes para la ejecución de una tarea no dependen de una temporización global, siendo distribuida entre los componentes.
- Independencia a fallos. Cada componente del sistema es independiente ante los fallos, permitiendo a los demás ejecutar sus tareas, manteniendo un continuo trabajo.

2.10 Cloud Computing

El cómputo en la nube es una colección de servicios sobre demanda, a los que se pagar por usar con base a las necesidades del usuario. Servicios de computo que hoy día abarca un gran mercado en las telecomunicaciones, debido a que libera al usuario de la inversión extra por adquisición y mantenimiento que podría implicar una infraestructura de computo de grandes proporciones, como son los clúster de supercómputo, servidores web, racks de almacenamiento masivo, etc.

Los servicios de cómputo en la nube se han convertido en un ecosistema complejo de cómputo distribuido comercial de nueva generación por su gran escalabilidad computacional para la implementación de diversas aplicaciones y creación de plataformas especializadas, directamente proporcionales a las necesidades del usuario/cliente.

Hoy día el Cloud Computing (computo en la nube) ha sido definido en dos tipos de nubes y divididas en tres servicios especializadas principalmente que permiten tener una mejor comprensión de su impacto y aplicación comercial.

- Nube pública. Administrada externamente por terceros, donde el contenido de varios usuarios puede encontrarse en los mismos servidores, sin que conozcan los trabajos de otros clientes y puedan estar trabajando en el mismo servidor.
- Nube privada. En esta nube el proveedor es propietario del servidor, la red, y discos de almacenamiento, se puede decidir qué usuarios están autorizados a acceder los servicios. Las nubes privadas están en una infraestructura manejada por un solo administrador que con-trola qué aplicaciones debe correr y dónde. Son una buena opción para las compañías que necesitan alta protección de datos y manipulaciones a nivel de servicio.
- Nube Hibrida. Múltiples nubes públicas y privadas, donde el cliente está en posesión de una parte y comparte otras. Perfecto para el escalado externo,

bajo demanda, se añade la posibilidad de determinar cómo distribuir las aplicaciones a través de los distintos ambientes.

La arquitectura de la computación en la nube se define estableciendo una separación de servicios ofrecidos para el consumo de hardware, plataformas y aplicaciones. Estas capas son:

- SaaS (Software as a Service). Esta capa del cómputo en la nube, software como servicio, permite al usuario correr aplicaciones enteras directamente desde la nube.
- PaaS (Platform as a Service). Se define por proveer un entorno de escritorio (virtualidad) con sistema operativo (SO) y los servicios que el usuario requiera para ejecutar y/o desarrollar una aplicación en específico.
- IaaS (Infrastructure as a Service). Este modelo se define por proporcionar un acceso a recursos informáticos, principalmente hardware, situados en un entorno de suministro de energía ininterrumpido, controles de acceso con facilidades de comunicación a los componentes y se enfoca en garantizar una infraestructura con poder de procesamiento y reservas de ancho de banda [30].

2.10.1 Beneficios

1. Ahorro: Un significante ahorro en la compra de licencias, administración de servicio y adquisición de hardware cómputo.
2. Implementación: Se reduce el tiempo de puesta en marcha de una plataforma o servicio en la nube, y reduce los riesgos que esto implica.
3. Portabilidad de la Información: El acceso a la información almacenada en la nube agiliza el modo de trabajo de las personas y empresas.
4. Ecológico: El uso de los servidores permite al usuario adquirir computadoras más austeras que consuman menos energía, y en cuanto a los servidores,

aunque estos consuman mucha energía, sirven las necesidades de muchos usuarios a la vez.

2.10.2 Desventajas

1. Fuga de información: Es evidente que el manejo de una gran variedad de la información que los proveedores almacenan podría evocar en una fuga de información, con un significativo impacto para el usuario. Usar los ser-vicios implica tener una confianza casi absoluta en el proveedor, dejando información importante en sus manos, que podrían ser objeto de ataques informáticos.
2. Disponibilidad sujeta a internet: El usuario adopta la necesidad de un punto de acceso a internet para consumir los servicios, viéndose su productividad posiblemente mermada o completamente nula.
3. Escalabilidad a largo plazo: El proveedor probablemente no contemple un crecimiento en el uso de su infraestructura, pudiendo generarse una sobre carga en los servidores, pudiendo llevar a degradaciones en el servicio. (Computación en la nube, 2011).

2.11 Grid

A inicios de los años 90’s se empezó a notar una evolución en los componentes tecnológicos que la investigación científica utilizaba, y las necesidades de los investigadores para poder continuar sus estudios, marcaron la pauta a un nuevo concepto de colaboración entre universidades y centros de investigación especializado, donde la necesidad de acceder a recursos de cómputo era cada vez mayor para realizar simulaciones realistas y análisis de grandes cantidades de datos. Siendo evidente de que no todas las organizaciones podían costear la adquisición de súper computadoras o data centers de gran envergadura, así como

el desarrollo de software especializado y con el subyacente reto de encontrar la manera de dar acceso a estas infraestructuras y demás instrumentación científica a quienes la necesitasen.

Compartir estos recursos heterogéneos es el principal objetivo para el desarrollo de un ecosistema interconectado con el cual proporcionar, a los investigadores o miembros de alguna organización, los recursos que requieren para poder realizar sus investigaciones.

A sí es como Ian Foster del Laboratorio Nacional de Argonne en Estados Unidos y Carl Kesselman de la Universidad del Sur de California, concibieron la idea de interconectar los recursos de las organizaciones colaborativas; universidades, laboratorios, centros de investigación especializado, etc. A este concepto le nombraron ‘computing grid’ (cómputo en malla), el término ‘grid’ fue tomado de ‘power grid’; siendo extensas estaciones generadoras de energía interconectadas para dar suministro eléctrico a varios consumidores.

Ian Foster y su equipo iniciaron un proyecto llamado Globus, del cual se hablará en un capitulo posterior. Este proyecto fue un esfuerzo colaborativo entre las organizaciones interesadas en la implementación de cómputo en malla, el cual ha sido definido de muchas maneras con el paso del tiempo gracias a su continuo desarrollo, siendo la definición más acertada la siguiente:

¨El cómputo grid permite el uso de un conjunto de estándares abiertos y protocolos que den acceso a aplicaciones y datos, poder de procesamiento, capacidad de almacenamiento, y gran cantidad de recursos de cómputo sobre internet. Es un tipo de sistema distribuido y paralelo que da disponibilidad a compartir, seleccionar e integrar múltiples recursos descentralizados en función de su disponibilidad, capacidad, rendimiento, costo y calidad de servicio requerido por los usuarios¨ [26].

Este concepto permitió dar un nuevo enfoque a los servicios sobre internet, siendo considerado predecesor al “Cloud Computing” y sus variados servicios, mencionados en el tema anterior, de entre estos su semejante actual a sería el

“Community Cloud”; variante comercial del Grid Computing, conceptualmente parecido a “Infrestructure as a Service” en el cual se da acceso a una infraestructura pero en este caso solo para uso específico de una comunidad de usuarios que comparten un mismo interés; como por ejemplo universidades o varias empresas que busquen interconectan sus recursos de computo como una nube privada, donde solo los miembros autorizados tienen acceso a estos. Así que estos recursos le pertenecen a la comunidad, pudiendo ser administrada por cada institución participante o por terceros autorizados.

Pero con la notable diferencia a un community cloud; donde solo se busca que todos estos recursos sean encapsulados para una organización bien definida, aspecto que en una grid no solo implementa, si no que va un poco más allá, buscando un ecosistema de cómputo distribuido expandido, si así se deseará, en concepto de procesamiento computacional, integrando el potencial de múltiples supercomputadores y computadores personales en un único supercomputador, virtualmente disponible para realizar tareas realmente pesadas, que a diferencia de los cloud que trabajan en un ecosistema para la WWW (World Wide Web), proveen modos básicos de trabajo conjunto, pero que aún no vincula y comparte computadores, sensores y demás recursos heterogéneos geográficamente dispersos, creando una única entidad virtual, como es el caso para el Grid Computing que si lo permite, aprovechando la sinergia que surge de la cooperación entre recursos computacionales compartidos, coordinarlos para el uso de las VO y proveerlos como servicios remotos.

2.11.1 Arquitectura

Una Grid debe contar con los siguientes componentes fundamentales del sistema junto con sus principales funciones que deberán cumplir las Toolkits APIs (Application Programming Interface) que se implementen para su construcción.

Esta arquitectura es abierta, donde los componentes del sistema se organizan en cinco capas Infraestructura, Conectividad, Recurso, Colecciones y Aplicación. Esta arquitectura permite a los usuarios gestionar los recursos compartidos.
En cada capa se admiten componentes de software específicos dependiendo de la capa que le preceda, en otras palabras los componentes de la capa de aplicación, dependerán de los servicios que se pretendan ofrecer en las capas inferiores.
En esencia la gestión y administración de esta arquitectura para este proyecto en particular vendrá de la mano del middleware Globus toolkit.

- La capa de Infraestructura son los componentes computacionales que serán compartidos por la organización virtual; Bases de datos, librerías de software, redes, HPC (High Performance Computing), instrumentación, etc.
- La capa de Conectividad contiene los protocolos estándar de seguridad y comunicación para transacciones de red. Los protocolos de comunicación permiten el intercambio de datos entre capas, mientras que los protocolos de seguridad ofrecen mecanismos de cifrado para identificar usuarios y recursos. Como se mencionó en apartados anteriores los protocolos de comunicación corresponden a TCP/IP y protocolos SSL y certificados X.509 para ofrecer la seguridad y autenticación necesaria.
- El Recurso corresponde a los protocolos que permiten obtener la información de un recurso en particular para ser gestionado bajo un control de acceso, arranque de procesos y monitorización QoS (Quality of Service).
- La capa de Colecciones contiene los protocolos y servicios que se tienen permitidos, y así poder gestionar un conjunto de recursos, como lo directorios, ubicación de recursos compartidos disponibles, administradores distribuidos, para asignar tareas a cada recurso.
- En la capa de Aplicación, se encuentran los módulos que permiten el acceso a la estructura Grid, pudiendo ser estas aplicaciones establecidas bajo términos de un servicio en específico, para alguna de las otras capas, pudiendo acceder a estas directamente.

Es importante enfatizar que en el Cloud Computing y Grid Computing son dos paradigmas distintos, aunque compartan similitudes, que hoy en día siguen vigentes con sus respectivos fundamentos, por ejemplo, a nivel de negocios:

2.12 Grid Cloud

Principalmente apoyadas por una entidad Gubernamental.
Apoyada por recursos de la instruía. Orientado a miembros institucionales, de gobierno y laboratorios de investigación científica. Orientado al usuario común, pequeñas y medianas empresas, con alguna incursión a la investigación independiente.
Orientado a proveer acceso a recursos para la investigación. Orientado a los negocios y servicios bajo demanda.
Por lo que se puede intuir que existen semejanzas entre los distintos sistemas distribuidos conocidos, pero con objetivos bien definidos, directamente relacionados a las capacidades y alcances que a cada infraestructura pudiera proveer, definiendo así una orientación hacia lo aplicativo o como en el caso del cloud computing, enfoca en proveer solo servicios bajo demanda.

Software que comprende las necesidades para dar soporte a la interacción y compartición de recursos entre las partes de un sistema distribuido, permitiendo establecer un enlace a los servicios que un cliente requiera de un servidor dentro de una infraestructura distribuida.
Implementada como una capa más en los modelos de telecomunicación a nivel de aplicación que oculta la heterogeneidad y provee de un modelo computacional uniforme.

Los tipos de middleware son:

- Intermedio General – Son los servicios que requiere todo cliente y servidor, permitiendo dar soporte a comunicaciones mediante TCP/IP, manejo de almacenamiento de archivos distribuidos, software de autenticación, etc.
- Intermedio de Servicios – Este está asociado a servicios en particular, como es el caso de CORBA que permite que objetos distribuidos creados en distintos lenguajes puedan coexistir en una misma red o software asociado a la seguridad especifica como la comunicación mediante Sockets.

Para desarrollar un ecosistema de comunicación segura Open Grid Forum (OGF) implementa estándares para el desarrollo y la implantación de arquitecturas de computo Grid, siendo la Globus Alliance la desarrolladora del el software Globus Toolkit, para la construcción de servicios Grid.

La OGF nos provee de un estándar abierto que comprende las 5 capas elementales de arquitectura Grid, para así poder establecer una comunicación con los recursos perteneciente dentro del Grid, es necesario entender la 'Política de Acceso' que comúnmente es establecida con un login y password, como en cualquier otra Web Services (WS), pero en un ecosistema Grid, es complementado utilizando la Grid Security Infrastructure (GSI), la cual está basada en criptografía de llaves públicas como RSA y certificados X.509 (Ian Foster C. K.), un estándar para codificar credenciales dentro de una infraestructura de certificación que expida certificados uniformes con los permisos necesarios para todos los miembros de la red.

Encontrándose en la etapa de la Autenticación y Autorización, una Grid tiene políticas de seguridad, que consiste en satisfacer las siguientes etapas:

Por consiguiente, al tener definidos los Objetos (servicios) y validar el acceso al Sujeto que envía la petición, se establece un canal directo al host servidor de la VO que tenga dicho objeto/servicio solicitado. En este proyecto se hace referencia al control y monitoreo de los DF como el servicio a consumir.

2.13 Organización virtual

La organización virtual se refiere a una estructura que promueve la realización de alianzas temporales entre personas, instituciones y empresas con el propósito de realizar tareas específicas enlazadas mediante el empleo de las tecnologías de la información, siguiendo esquemas que reducen los costos y facilitan el acceso a las capacidades y recursos de las partes involucradas.

Comúnmente estructuradas en organizaciones muy bien definidas, donde cada VO es administrada por uno o más usuarios, con los privilegios y responsabilidades de definir la estructura de la VO (grupos, roles, atributos, clases), aprobar las peticiones de usuarios y gestionar tareas administrativas.

Un ejemplo es ‘atlas’ una VO del CERN (Conseil Européen pour la Recherche Nucléaire) que es la Organización Europea para la Investigación Nuclear, que tiene 3000 usuarios registrado en 70 grupos y entre otras VO pertenecientes al GRID del CERN [29] .

3 Formulación, Planeación y Análisis

3.1 Introduccion

Una computadora sin software es sólo una masa metálica sin utilidad, y de manera inversa un software será un conjunto de instrucciones lógicas sin un espacio donde aplicarse. Entre las funciones principales de una computadora se puede mencionar el almacenamiento de datos, procesamiento y recuperación de información; una computadora está compuesta de una estructura física llamada hardware que puede representarse de forma abstracta con los siguientes componentes o recursos.

El proceso es el concepto central de los Sistemas Operativos. El proceso puede de verse como la abstracción de un programa en ejecución y cada uno tiene su propia CPU virtual de manera conceptual. En un Sistema Operativo multicapas (Unix) el proceso se crea mediante la llamada al sistema fork(). Esta llamada al sistema hace que el núcleo del sistema operativo (kernel) realice los siguientes 5 puntos:

1. El kernel busca una entrada libre en la tabla de procesos y la reserva para el proceso hijo.
2. Asigna un identicador de proceso (PID) para el proceso hijo.
3. Este número es único e invariable durante toda la vida del proceso y también es la clave para poder controlarlo desde otro proceso o en una terminal.
4. El S.O. realiza una copia del contexto a nivel usuario esto para reservar un espacio en memoria y es asignada del proceso padre al proceso hijo. Las secciones que deben ser compartidas como el código o segmentos de memoria no son copiadas, por el contrario se incrementan los contadores que indican el número de procesos que comparten dicho segmento.
5. Las tablas de control en el fichero local, que se encuentra en el proceso (como pueden ser la tabla de descriptores de archivos o fichero), también se hereda del proceso padre al proceso hijo, ya que forman parte del contexto

a nivel usuario. En las tablas globales del núcleo (tabla de ficheros y tabla de i-nodos) se incrementan los contadores que indican cuantos procesos tienen abiertos los cheros. Así también, el Kernel retorna el PID del proceso padre al proceso hijo, y al proceso hijo le devuelve el valor 0.

Como sabemos, las computadoras pueden llevar a cabo diversas tareas a la vez, por una parte puede realizar lecturas a discos y casi al mismo tiempo utilizar una impresora. Esto puede realizarse ya que la CPU ejecuta en cierto instante un sólo programa, durante un lapso de tiempo puede trabajar con varios procesos que realizan tareas simultáneamente, lo que da una apariencia de paralelismo, pero al tratarse de una CPU tiende a la concurrencia como se ilustra en la figura 1.

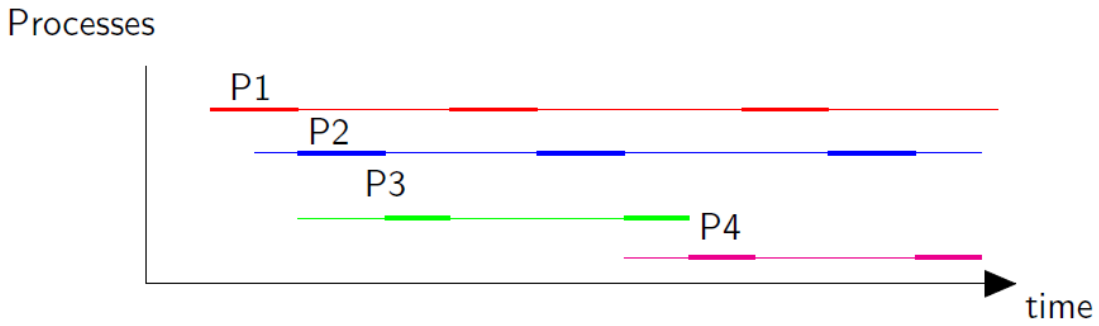

Figura 1. Multiprocesos

La diferencia entre la programación concurrente y la programación paralela se basa en el número de recursos en una computadora en la concurrencia y el número de recursos en varias computadoras en el caso del paralelismo, de esta forma podemos definir la concurrencia y el paralelismo como se sugiere en [27].

Definimos a la programación concurrencia y a la programación paralela [25] como:

Concurrencia: Cuando dos o más procesos pueden estar ejecutando simultáneamente una parte de código en una CPU.

Paralelismo: Cuando dos o más procesos pueden estar ejecutando simultáneamente una parte de código en dos o más CPU como se observa en la figura 2.

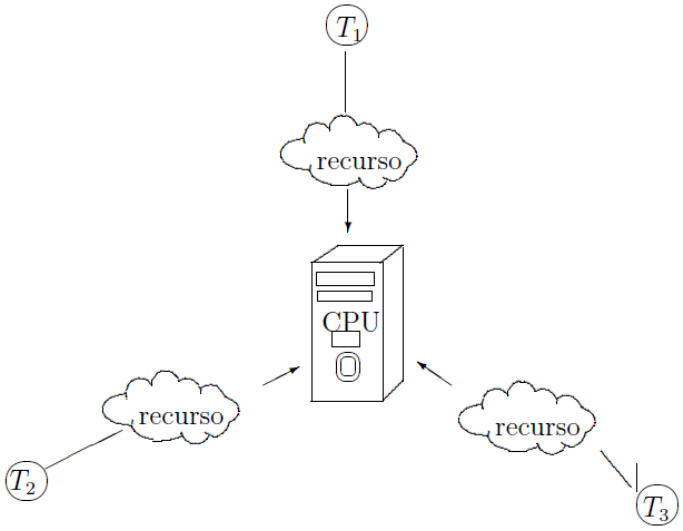

Figura 2 Concurrencia

Tomando en cuenta el ejemplo anterior, además de contemplar el comportamiento del kernel de los sistemas operativos y aplicaciones de software, podemos definir algunos conceptos como:

Definición. Condiciones de competencia. Cuando dos o más procesos leen o escriben en ciertos datos compartidos y el resultado final depende de ¿quién ejecute qué? y en ¿qué momento?

Si un proceso utiliza una variable V o un archivo A compartidos, los demás procesos no pueden utilizarlos, ya que esto podrá tender a las condiciones de competencia. A este concepto se le da el nombre de **Exclusión Mutua**.

Una vez que definimos la exclusión mutua, es importante mencionar que los procesos no siempre están compitiendo por la sección critica (SC), también realizan cálculos o procesan información que no conducen a las condiciones de competencia o a utilizar un recurso que sea requerido por otro proceso, sin embargo otros procesos comparten recursos o realizan labores críticas que pueden llevar a conflictos.

El bloqueo (**mutex**) es la variable de sincronización más simple y primitiva. La regla básica es que el primer proceso que bloquee el mutex obtiene sus propiedades y los demás procesos no tendrán éxito si su estatus está en espera hasta que el proceso que se encuentra en su SC salga de ella. Existe la posibilidad de que algún otro proceso bloque el mutex y obtenga la propiedad antes de que lo haga el que se acaba de activarse, por prioridad. Por ejemplo supongamos que se tiene tres Procesos T_1, T_2 y T_3. Los tres procesos necesitan un mutex con diferentes prioridades P_1, P_2 y P_3 respectivamente, que determinan el orden en que pasan a la cola de espera. El thread T_1 se queda con la posición del bloqueo, y el thread T_2 solicita el bloqueo antes que el thread T_3, este se reactiva en cuanto el thread T_1 libere dicho bloqueo. Por lo cual se tiene un problema de sincronización que el mutex no logra resolver.

Es el acto de extraer un Proceso activo de su SC y sustituirlo por otro que está esperando para ejecutarse se le conoce como **conmutación de contexto** y una forma de que el mutex pueda realizar la sincronización, aunque esto se realiza más para sistema de tiempo real hard. Esta operación es crítica en la ejecución de instrucciones en una CPU. Cuando se realiza una conmutación de contexto, se debe de guardar información importante de los registros, pilas, apuntadores de segmentos entre otros. Los datos importantes que se guardan en una conmutación de contexto son los siguientes:

- Contador de programa
- Puntero de pila
- Registros generales
- MMU(unidad de gestión de memoria)
- Contenido de la memoria

Cuando llega el momento de conmutar el contexto de dos procesos tradicionales, hay que cambiar el estado del registro para que retome los nuevos procesos que

queremos ejecutar. Todos los registros activos se almacenan en la estructura del proceso P_1. Todos los valores almacenados del registro de la estructura del proceso P_2 se cargan en los registros de la CPU. Y por último la CPU vuelve al modo usuario. El proceso P1 se ha conmutado de contexto y el proceso P2 se ha introducido en contexto y está funcionando. La conmutación de contexto debe ser realizada por la propia CPU. Una CPU no puede llevar a cabo las conmutaciones de contexto de otra CPU, a su vez la CPU_1 puede enviar una interrupción a CPU_2 y la CPU_2 debe estar de acuerdo en conmutar los contextos. Cuando se realiza una operación de esta forma, es importante comparar las prioridades de los procesos puesto que la conmutación de contexto sólo se realiza cuando los procesos tienen las mismas prioridades, en caso contrario, se debe realizar antes una operación para igualar dichas prioridades. Utilizando mutex se puede realizar la conmutación de contexto, sin revisión de prioridades, ya que mutex no garantiza sincronización bajo prioridades de procesos.

3.2 Peers

El peer to peer, también conocido como P2P, red entre pares, entre iguales o punto a punto, es una red de computadoras en la que todos los aspectos funcionan con una serie de nodos que se comportan de la misma manera entre sí y sin clientes ni servidores fijos. De esta manera se intercambia de forma directa cualquier tipo de información entre los ordenadores que estén interconectados..

Crear varios peer y que todos estos tengan un identificador como la fecha y hora que fueron creados, estos serán creados por java y c++ y puedan conectarse para comunicarse sin tener un servidor central.

3.2.1 La Interface Runnable.

Se tiene una clase que implementa la interfaz Runnable y un método run(), luego se crea un thread con esta interfaz como argumento y se llama al método start(). Las tareas se ejecutan en forma concurrente.
El ejemplo se puede ver en la figura 3.

```
public interface Runnable {
        public void run( );
}
public class MyRunnable implements Runnable{
        public void run(){
            // Tarea que realizará el hilo
}
MyRunnable r=new MyRunnable()
thread t = new Thread(r).start();
```

Figura 3. Interface Runnable

La razón principal para usar una interfaz Runnable es que no se modifica la naturaleza de la clase Thread, es decir, se utiliza un método llamado run() que el usuario podrá implementar. También es posible crear una subclase por cada thread.

1. public class Thread extends Object implements Runnable
2. public Thread(Runnable ObjetoEjecutable)
3. new Thread()public void run()doWork();.start();

3.2.2 Modos de usar la Interface Runnable

Existen varias formas de crear y ejecutar a los threads. Las tres formas principales se exponen en el código anterior, donde la primera línea (1) nos indica que se crea

un Thread con herencia de la clase objeto e implementación de la interface Runnable. Esto nos indica que tenemos todas las propiedades de la clase madre Object, pero solo podemos implementar el método run() sin tener directamente las propiedades de la clase Thread. En la línea (2) se crea la interface Runnable en el constructor del Thread o se puede construir antes el objeto y enviarlo por medio del constructor. La línea (3) es más abstracto el mecanismo, pues se crea el thread, a la vez que se implementa el método run() en el constructor y por último se ejecuta el thread con el método start() [31].

3.2.3 Thread deamon

Un Thread Daemon es un thread que se ejecuta en beneficio de otro thread. Se ejecutan en un segundo plano (cuando hay tiempo de procesador disponible). El recolector de basura es un hilo se puede ver en la figura 4.

```
Daemon. P/E
setDaemon(true);
if(isDaemon)//El Thread es un Daemon
```

Ilustración 4. Métodos de un thread daemon

Existen dos tipos de Threads: Thread Deamon y Thread usuario. Los threads deamons son creados internamente por la máquina virtual y los threads usuario son creados por el programador. La única diferencia entre estos dos tipos de threads es que solo se ejecutan a nivel servidor. El modo Deamon de un thread se agrega a un conjunto realizando la llamada al método setDaemon(). Este método solo puede ser llamado por un método inicializado, si el thread ya está ejecutándose ya no puede ser un Thread Daemon (o viceversa).

3.2.4 Grupo de threads.

Los grupos de threads se definen en la clase java.lang.ThreadGroup. En un grupo de threads siempre existe un thread principal llamado *thread main* del grupo". También existen grupos de threads padre y grupo de threads hijo siempre y cuando exista una jerarquía de Grupos de Threads como se muestra en el diagrama siguiente:

El método getThreadGrouop() es instanciado para unir un thread a un grupo. Las ventajas del uso de grupos de threads son dos:
La clase de grupo de thread permite operar a todos los threads del grupo, es decir, se puede interrumpir a un grupo de threads o a un solo thread.
La seguridad en los grupos de threads es adecuada en las aplicaciones y en la decisión ordenada en el acceso y/o modificación de los estados de los threads que se encuentran en los grupos.
La forma de crear un grupo de threads es heredado de la clase ThreadGroup como se muestra en [32]:
Los métodos de excepciones para la clase ThreadGroup son:

1. setDefaultExceptionHandler(Thread.UncaughtExceptionHandler). El llamado de este método nos indica que el constructor ya ha sido construido y obtiene el error para no abortar la aplicación.
2. setExceptionHandler(Thread.UncaughtExceptionHandler). Por default la clase ThreadGroup toma esta excepción para evitar errores y que la aplicación se recupere.

3.3 Paradigmas de la comunicación remota

En la concurrencia como en el paralelismo la comunicación entre dos o más CPU's es fundamental, a pesar de que por definición la concurrencia no necesita más que un CPU, las aplicaciones de hoy en día y en un futuro son y serán de forma remota, basadas rede Web e inalámbricas. El principio de toda aplicación se remonta a una

CPU, luego en dos o más CPU´s hasta llegar a clúster LAN y WAN como se muestra en la figura 5.

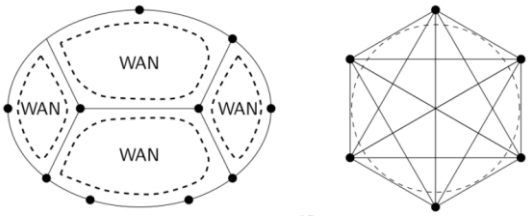

Figura 5. Redes LAN WAN

En este capítulo se exponen tres métodos de comunicación remota los cuales son: SOCKETS, RMI1 y CORBA2. Además de tres importantes paradigmas de comunicación entre computadoras, por ejemplo: Maestro/esclavo, cliente/servidor y peer to peer (participante/participante) [33].

3.3.1 Maestro-Esclavo

Este paradigma está orientado a una comunicación básica entre dos o más computadoras, se puede decir que es la forma primitiva en que se comunican dos computadoras. La filosofía del Maestro-Esclavo es muy semejante al de Humano-Computadora. El funcionamiento del Maestro-Esclavo consiste enviar datos a los esclavos, luego de establecer la relación Maestro-Esclavo y la jerarquía de la dirección del control del programa. La única comunicación que se tienen entre maestro y esclavo es enviar los resultados de la tarea asignada, comúnmente no existen dependencias fuertes entre las tareas realizadas por los esclavos (poca o nula comunicación entre esclavos).

La sincronización y la asincronización se realizan de forma sencilla pero requiere programar el mecanismo de lanzado de tareas, la distribución de datos, el control del maestro sobre los esclavos y la sincronización en caso de ser necesaria.

Las aplicaciones típicas de este paradigma son:

- Técnicas de simulación Monte Carlo.
- Aplicaciones criptográficas.
- Algoritmos de optimización (basados en poblaciones).
- Modelos de partición sencilla de tareas y datos.
- Otros modelos del paradigma Maestro-Esclavo como:

Los Modelos fork-join y FORK L, generan dos ejecuciones concurrentes en un programa. Una se inicia en la instrucción FORK y otra empieza en la instrucción etiquetada L. JOIN permite recombinar varias ejecuciones paralelas en una sola. La rama que ejecuta primero la instrucción JOIN termina su ejecución.

Para saber el número de ramas que se deben reunir se usa un parámetro con JOIN (una variable entera no negativa que se inicializa con el número de ejecuciones paralelas a reunir). La instrucción JOIN tiene que ejecutarse indivisiblemente es decir, la ejecución concurrente de dos instrucciones JOIN es equivalente a la ejecución secuencial en un orden indeterminado [27].

Algorithm Maestro

```
1: (num_esclavos);
2: for i=0; i=num_esclavos do
3:    datos=Determinar_datos(i);
4:    Lanzar_tarea(i, datos);
5: end for
6: respuestas=0;
7: for i=0; i=num_esclavos do
8:    res=Obtener_Respuesta();
9:    resultado=Procesar_resultado(res);
10: end for
11: Desplegar_resultado(resultado);
```

Algorithm Esclavo

```
1: datos=Esperar_datos(master);
2: Procesar(datos); {No hay comunicación entre pro-
   cesos esclavos }
3: Enviar_Resultado(master);
```

Figura 6. Algoritmo de Maestro-Esclavo

3.3.2 Cliente-Servidor

El paradigma cliente-servidor nos proporciona la forma fácil de comunicar dos computadoras, con la restricción de que dos clientes no podrán comunicarse entre ellos. Es decir, se da el nombre de servidor a una computadora y el nombre de cliente a otra, y estas dos se comunican por medio de mensajes. El cliente realiza peticiones de recursos de computo al servidor y el servidor atiende estas peticiones, la restricción se encuentra en que un cliente no puede hacer peticiones de recursos a otro cliente, la única forma de comunicación entre dos cliente es por medio del servidor(ver fig. 6).

En la figura 7 se muestra el flujo de comunicación entre varios clientes y un servidor y las respectivas restricciones que se tiene un el paradigma cliente-servidor.

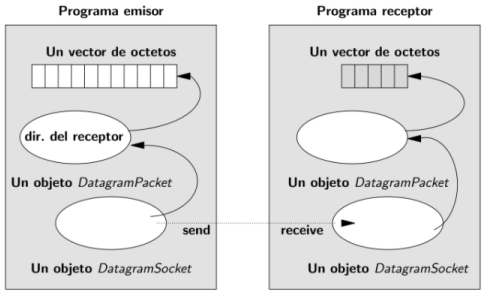

Figura 7. Comunicación Cliente-Servidor

Para que el servidor puede atender todas las peticiones de los clientes, se crean procesos o threads para cada uno de estos clientes conectados. Esto se realiza para que el CPU por completo, que no tenga que resolver cada una de las peticiones de recursos o servicios que los clientes realizan con frecuencia, dado una CPU abstracta o conceptualizada por cada uno de los clientes hace que se maximice el cómputo y minimice el tiempo de respuesta de recursos o peticiones.
La forma conceptual en que un servidor puede atender a varios clientes conectados y luego detectados por procesos o threads por medio de un canal de comunicación bidireccional. Este concepto es importante y básico en el paradigma Cliente-Servidor por lo mismo que el servidor.

Otra forma de expresar el paradigma Cliente-Servidor es estructurando un sistema operativo como un grupo de procesos cooperativos, llamados servidores, que ofrecen servicios a los usuarios llamados clientes. El servidor y el cliente normalmente se ejecutan sobre el mismo kernel o micro kernel en máquinas diferentes. El cliente y servidor se basan en un protocolo sin conexión de solicitud/respuesta como se ve en la figura 8.

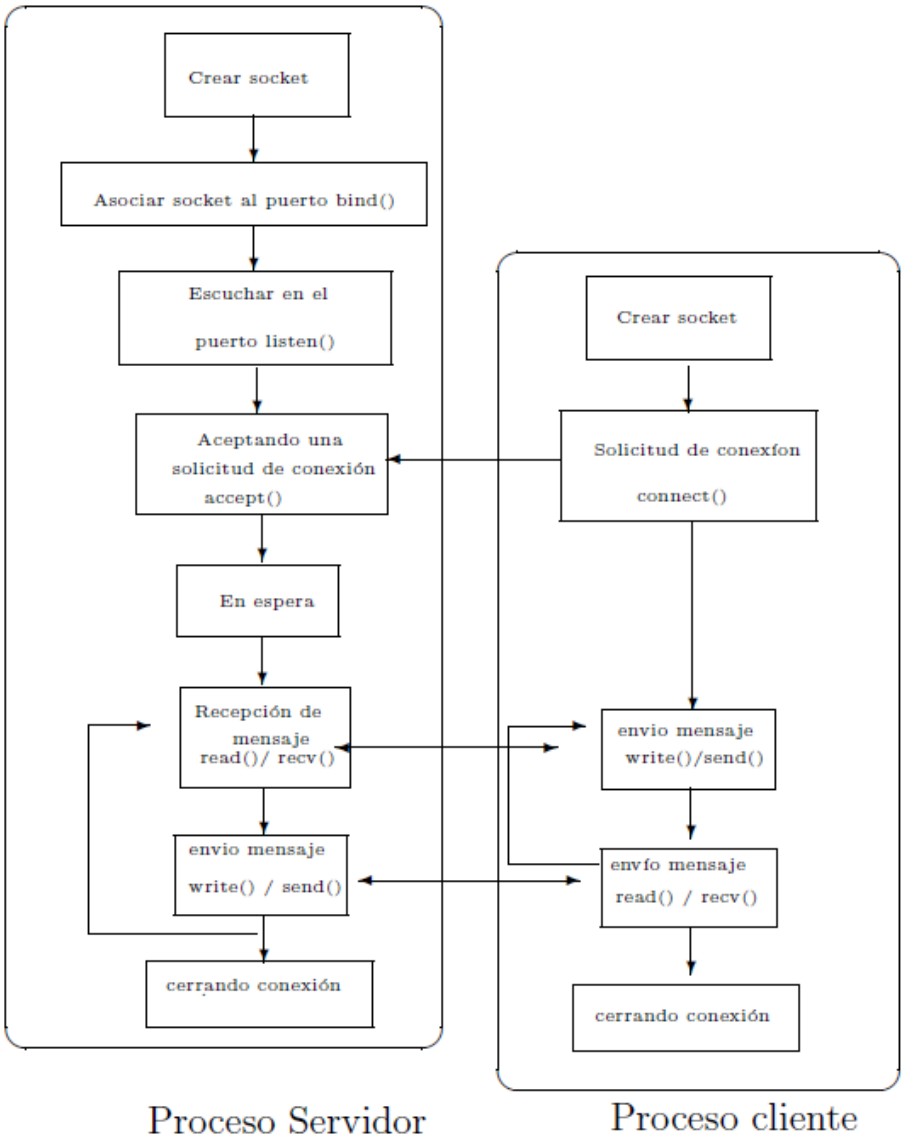

Figura 8. Cliente-Servidor

Las ventajas del paradigma Cliente-Servidor en primer lugar, está basado en un sistema simple, sostenidos en las tres primeras capas del modelo OSI 3. Proponiendo dos primitivas send y receive de la manera siguiente:

- send(dest,menssPtr)
- receive(addr,menssPtr)

Donde *dest* es igual a destino, *addr* la dirección quien envió y menssPtr el mensaje como un apuntador de memoria. Un cliente debe conocer la dirección del servidor al que le desea enviar un mensaje. La estrategia para realizar una comunicación entre cliente y servidor es:

1. Especificación de la máquina. Valido para un proceso en la máquina.
2. Especificar la máquina y el proceso. Las maquinas inician la numeración de sus procesos a partir de cero (no es transparente).
4. Especificar la máquina y un identificador local entero aleatorio de 16 a 22 bits.
5. Especificar el proceso. Definiendo un proceso centralizado que asigne direcciones únicas a los procesos que lo soliciten (no conveniente para grandes sistemas).
6. Permitir que los procesos seleccionen aleatoriamente su direccionamiento grande.

El modelo de comunicación que identifica dos clases de procesos, como puede ser una clase cliente que solicita servicios a procesos de otra clase servidor, que atiende los pedidos y puede ser utilizada para comunicar procesos que se ejecutan en un único equipo, a la vez es una idea potencialmente utilizada para comunicar procesos distribuidos en una red. El modelo Cliente-Servidor provee un mecanismo para comunicar aplicaciones remotas (que funcionen simultáneamente como clientes y

servidores para diferentes servicios, convenientemente de acuerdo a las características de la red.

3.3.3 peer-to-peer

El paradigma peer-to-peer (P2P) es abierto con respecto a las restricciones que tienen los paradigmas Maestro-Esclavo y Cliente-Servidor. En el paradigma p2p se puede comunicar un cliente con otro cliente y también se pueden pedir recursos entre estos.

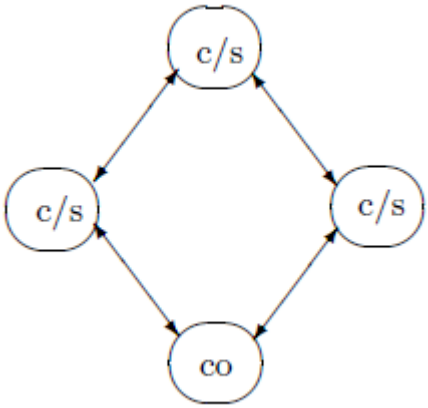

Figura 9. P2P

Se puede ser un cliente-servidor al mismo tiempo o también conocidos como peer. La nomenclatura ***co*** en la figura 9, es un posible coordinador entre los peers, el cual representa el nodo que contiene la información de entrada y salida de los demás nodos en la red, y cada vez que se realiza algún cambio este notifica a los demás peers. Con un grafo dirigido se puede representar una red p2p por que el flujo de datos es bidireccional en cada uno de los peers.

Una red informática peer-to-peer es un tipo particular de red en la que los usuarios y sus nodos son anónimos por defecto. La principal diferencia entre las redes habituales y las anónimas se encuentran en el método de encaminamiento de las respectivas arquitecturas de redes. Estas redes permiten el flujo libre de información. Y el interés de la comunidad p2p en el anonimato ha incrementado muy

rápidamente desde hace unos años por varias razones, entre ellas se encuentra la desconfianza en todos los gobiernos y los permisos digitales. Tales redes suelen ser utilizadas por aquellos que comparten ficheros musicales con copyright.

Muchas asociaciones referentes a la defensa de los derechos de autor han amenazado con demandar a algunos usuarios de redes p2p no anónimas.

El p2p se basa principalmente en la filosofía y los ideales, que señala que todos los usuarios deben compartir recursos conocida como filosofa p2p, es aplicada en algunas redes en forma de un sistema enteramente métrico en donde el que más comparta, mas privilegios tiene y más acceso dispone de manera rápida y con más contenido. Con este sistema se pretende asegurar la disponibilidad del contenido compartido, ya que de lo contrario no será posible la subsistencia de la red como se ve en la figura 10.

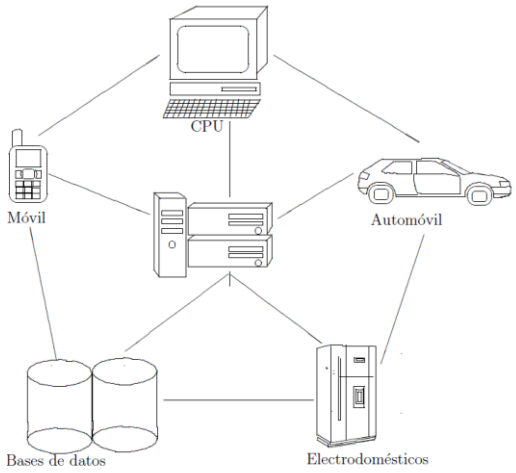

Figura 10. Ambiente distribuido

Características deseables de las redes p2p:

- Escalabilidad. Las redes p2p tienen un alcance mundial con cientos de millones de usuarios potenciales. En general, lo deseable es que cuantos más nodos están conectados a una red p2p mejor será su funcionamiento. As, cuando los nodos llegan y comparten sus propios recursos. Esto es diferente en una arquitectura del modo cliente-servidor con un sistema de servidores, en los cuales la adición de más clientes podrá significar una transferencia de datos más lenta para todos los usuarios. Algunos autores advierten de que si proliferan mucho este tipo de redes podrán llegar a su totalidad de conexión, ya que a cada una de estas redes se conectarán muy pocos usuarios.
- 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, y en sistemas p2p puros, permitiendo a los peers encontrar la información sin hacer peticiones a ningún servidor centralizado de indexado. En el último caso, no hay ningún punto singular de falla del sistema.
- Descentralización. Estas redes por definición son descentralizadas y todos los nodos son iguales. No existen nodos con funciones especiales, y por tanto ningún nodo es imprescindible para el funcionamiento de la red.
- Costes. Los costes están repartidos entre los usuarios. Se comparten o donan recursos a cambio de recursos, según la aplicación de la red, los recursos pueden ser archivos, ancho de banda, ciclos de proceso o almacenamiento de disco.
- Anonimato. Es deseable que en estas redes quede anónimo el autor de un contenido, el editor, el lector, el servidor que lo alberga y la petición para encontrarlo siempre que necesiten los usuarios. Muchas veces el derecho al anonimato y los derechos de autor son incompatibles entre estos, y la industria propone mecanismos como el DRM4 para limitar ambos.
- Seguridad. Es una de las características deseables de las redes p2p menos implementadas. Los objetivos de un p2p seguro serán identificar y evitar los

nodos maliciosos, evitar el contenido infectado, evitar el espionaje de las comunicaciones entre nodos, creación de grupos seguros de nodos dentro de la red, protección de los recursos de la red, entre otras. En su mayora aun están bajo investigación, pero los mecanismos más prometedores son:

Arquitectura Peer to Peer

Peer-to-peer es una arquitectura donde los recursos y los servicios son directamente intercambiados entre las computadoras, estos recursos y servicios incluyen el intercambio de información, los ciclos de procesamiento, almacenamiento en cache y el almacenamiento en disco para archivos; también el tipo de arquitectura, los equipos que han sido tradicionalmente utilizados únicamente como los clientes se comunican directamente entre sí pueden actuar como clientes y servidores, asumiendo cualquier papel que resulte más clientes para la red. En el paradigma peer-to-peer, los procesos participantes juegan un papel igual, con las capacidades y responsabilidades equivalentes. Cada participante podrá enviar una petición a otro participante y recibir una respuesta, mientras que el paradigma cliente-servidor es un modelo ideal para un servicio centralizado de la red, el paradigma peer-to-peer es más apropiado para aplicaciones como la mensajera instantánea, transferencias de archivos, video conferencia y el trabajo colaborativo. Por ejemplo un servicio de intercambio de archivos peer-to-peer de transferencia, como lo fue Napster.com, entre varios sitios similares que permiten que los archivos (archivos de audio sobre todo) se transmite entre las computadoras conectadas a Internet y puedan ser descargados sin derechos de autor.

3.3.4 Sistemas Distribuidos

Anteriormente la computación fue concebida mediante un salo procesador. La computación con una CPU llamada servidos, puede llamarse computación

centralizada y varias computadoras que comparten obligaciones y recursos se llama computación distribuida.

3.3.5 Definición Sistema Distribuido.

Es una colección de computadoras independientes conectadas va Red capaces de realizar trabajos colaborativamente. La computación distribuida es la computación que se lleva a cabo en un sistema distribuido, cuyo objetivo es el compartir recursos y llevar un control de usuarios y anónimos [35].

Ejemplos de sistemas distribuidos

Red de estaciones de trabajo (NOW): Un grupo de estaciones de trabajo en red conectadas a una o más máquinas de servidores.
Una Intranet. Una red de computadoras y estaciones de trabajo dentro de una organización, separados de la Internet a través de un dispositivo de protección (firewall).

Computadoras en un sistema distribuido

En la programación distribuida podemos contar con una larga lista de aplicaciones y tecnologías que involucran esta forma de pensar descentralizada y paradigmas P2P. Por ejemplo la figura 11.

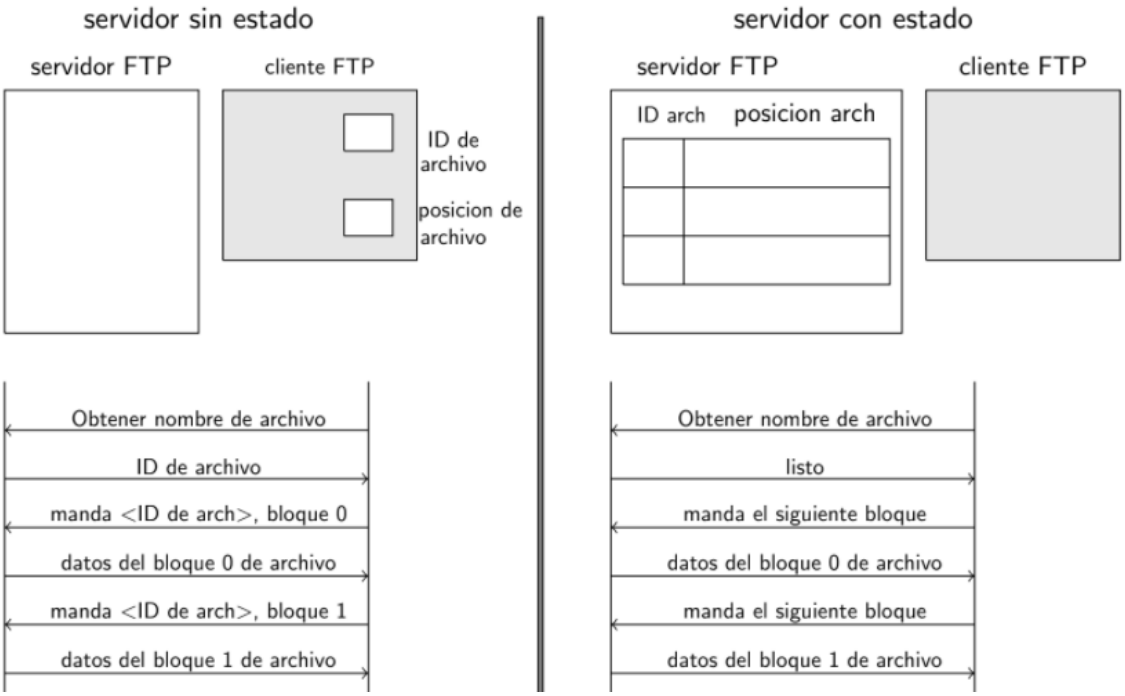

FIgura 11. Funcionamiento de un FTP

- Workstations: computadoras utilizadas por los usuarios para llevar a cabo el computo.
- Server: Computadoras que proporcionan recursos y servicios.
- Personal Assistance Devices (PDA): computadoras portátiles conectadas a la red a través de un enlace de comunicación inalámbrica.
- Aplicaciones monolíticas centrales vs Aplicaciones distribuidas.
- Arquitectura de las aplicaciones monolíticas: Independiente, de una sola función en la aplicación, tales como el orden de entrada o de facturación. Las aplicaciones no pueden compartir los datos u otros recursos. Los desarrolladores deben crear varias instancias de la misma funcionalidad (servicios).
- Arquitectura de las aplicaciones distribuidas.
- Aplicaciones integradas.
- Las aplicaciones pueden compartir recursos.

- Una sola instancia de la funcionalidad (servicios) pueden ser reutilizados.
- Interfaces de usuario común.
- Evolución de los paradigmas
- Cliente-Servidor. Este paradigma se puede implementar con métodos como: Sockets, métodos de invocación remota, entre otros.
- Objetos Distribuidos
- Servicios de Red : Jini
- Espacio de Objetos: Java Spaces
- Agentes móviles
- Mensajes orientados a middleware (MOM):Java MessageService
- Aplicaciones colaborativas.

¿Por qué computación distribuida?

- Económica: Los sistemas distribuidos permiten el uso común de recursos, incluidos los ciclos de CPU, almacenamiento de datos de entrada / salida, y los servicios.
- Fiabilidad: Un sistema distribuido permite la replicación de los recursos y / o servicios, reduciendo la interrupción del servicio debido a fallas. La Internet se ha convertido en una plataforma universal para la informática distribuida.

Debilidades y fortalezas de la computación distribuida En cualquier forma de cómputo, siempre hay ventajas y desventajas, algunas por razones de popularidad de la computación distribuida son:

- La accesibilidad de las computadoras y la disponibilidad de acceso a la red.
- Distribución de recursos.
- Escalabilidad.
- Tolerancia a fallas.
- Las desventajas de la computación distribuida:

- Múltiples puntos de error: Una o más computadoras o uno o más enlaces de red.
- Seguridad: En un sistema distribuido existen más oportunidades de un acceso no autorizado.

3.4 Sockets

Protocolo de comunicación remota. Un protocolo es un conjunto de reglas, formato de datos y convenciones que es necesario respetar para conseguir la comunicación entre dos entidades, ejemplos: IP, TCP, UDP, FTP, etc.1, Cuando se crea un socket es necesario indicar cuál es el protocolo de trasporte a emplear para el intercambio de datos que se realizara a través de él.

Es un mecanismo que comunica a más de dos procesos, que pueden intercambiar diferente tipos de datos ya sea que los procesos estén corriendo en un computador o en una red de computadoras, además guarda cierta similitud con las tuberías (pipe). Para que la comunicación se establezca entre los procesos son necesarios los siguientes aspectos:

- Dirección IP
- Protocolo
- Numero de puerto.

Para conseguir esto, tomamos el concepto de conector o socket; dos procesos distintos crean cada uno su conector por lo tanto.

1. Cada conector está ligado a una dirección.
2. Un proceso puede enviar información, a través de un socket propio, al socket de otro proceso, siempre y cuando conozca la dirección asociada a otro socket.

3. La comunicación se realiza entre una pareja de sockets.
4. Dominios y direcciones

Definimos un dominio de comunicación como una familia de protocolos que se pueden emplear para conseguir el intercambio de datos entre sockets. Un sistema UNIX particular puede ofertar varios dominios, aunque los más habituales son estos dos:

1. Dominio UNIX (PF1 UNIX). Para comunicarse entre procesos dentro de la misma máquina.
2. Dominio Internet (PF INET) Para comunicarse entre procesos en dos computadoras conectadas mediante los protocolos de Internet (TCP, UDP, IP).

Además un socket del dominio PF UNIX no puede dialogar con sockets del dominio PF INET. Cada familia de protocolos define una serie de convenciones a la hora de especificar los formatos de las direcciones. Tenemos, por lo tanto, estas dos familias de direcciones:

1. Formato UNIX (AF2 UNIX) Una dirección de socket es como los nombres de los ficheros (pathname).
2. Formato Internet (AF INET) 3. Una dirección de sockets precisa de estos tres campos:
    a. Una dirección de red de la maquina (Dirección IP).
    b. Un protocolo (TCP o UDP) y Un puerto correspondiente a ese protocolo.

Es importante resaltar que un socket puede ser referenciado desde el exterior solo si su dirección es conocida. Sin embargo, se puede utilizar un socket local, aunque no se conozca la dirección que tiene. Estilos de comunicación orientados a conexión

mecanismo que sirve para conseguir un canal para el intercambio de datos. Un proceso pone a su ritmo bytes en el canal, que recibe de forma fiable y ordenada en el otro extremo donde hay un proceso que los recoge a su conveniencia como se muestra en la figura 12.

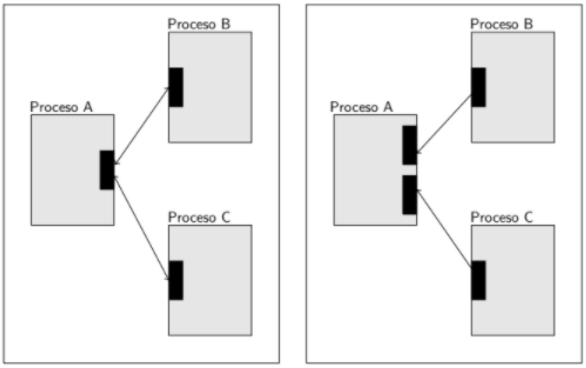

Figura 12. Socket orientado a sin conexción

Ejemplo: tuberías y socket pairs.

Comunicación sin conexión también denominada comunicación con datagramas. El emisor envía mensajes autónomos. Un mensaje autónomo es aquel que el receptor debe recibir entero por algún camino. Los mensajes pueden perderse en el camino, retrasarse o llegar desordenados. La comunicación entre dos sockets puede realizarse con cualquiera de estas dos formas (aunque los dos sockets que se comunican entre ellos deben de crearse con el mismo estilo, además del mismo dominio). Para ello, al crear un socket se indica el estilo de comunicación correspondiente:

SOCK STREAM La comunicación pasa por tres fases:

1. Apertura de conexión
2. Intercambio de datos
3. Cierre de conexión

El Socket es un concepto abstracto por el cual dos programas (posiblemente situados en CPU distintas) pueden intercambiar cualquier flujo de datos, generalmente de manera fiable y ordenada. Los sockets son una forma de

comunicación de computadoras que funcionan como si se conectara una computadora a otra por medio de un cable. Los sockets se dividen en dos tipos, (como se muestra en la figura 15):

- UDP (User Datagram Protocol)
- TCP (Transmision Control Protocol)

El Socket UDP transmite paquetes individuales de información. Los Sockets UDP son muy diferentes a los TCP, pues los UDP dan servicio sin conexión, pero no garantizan que los paquetes llegan en alguna forma en particular. El problema de los UDP es que los paquetes pueden perderse, duplicarse o llegar en desorden, es por eso que se requiere una programación adicional significativa para la solución de este problema.
Los servicios sin conexión generalmente ofrecen mayor velocidad, pero menor contabilidad que los servicios orientados a conexión (ver fig. 13).

Los APIs Socket en Java contienen dos tipos de clases para los UDP:

- DatagramSocket. Intercambio de datos
- DatagramPacket. Intercambio de datagramas

Figura 13. Izquierda TCP, derecha UDP

El socket TCP o también llamados socket de flujo, emplean un proceso que establece una conexión de un proceso P_A a un proceso P_B u otros procesos y que

esta comunicados y se sostiene mientras exista la misma conexión, de envió de datos y fluyen entre los procesos en un continuo flujo, proporcionando un servicio orientado a conexión, empleando un protocolo para la transmisión popular. El flujo donde pasan los mensajes en una comunicación con sockets datagrama orientado a conexión es mediante un camino virtual y los mensajes son estrictamente enviados de forma sincronizada y ordenada [35].

4 Desarrollar la plataforma embebida que soporte el análisis de un sistema distribuido móvil.

En este algoritmo propuesto en la figura 14, destaca el manejo, creación y escalamiento de procesos con UDP.

Algorithm 62 Conexión de sockets usando procesos

```
1: #define men1 "papá esta ahi?"
2: #define men2 "aqui estoy hijo"
Require: int n,sockets[2], child;
Require: char buf[1024];
3: if socketpair(PF_UNIX,SOCK_STREAM,0,sockets)<0 then
4:    perror("abriendo el par de sockets");
5:    exit(1);
6: end if
```

Figura 14. Creación de procesos

Después se aplica un deadline específico como se ve en la figura 14. Los procesos tienen la tarea de enviar mensajes en un tiempo determinado, creando un proceso por mensaje, de esta forma se mide el rendimiento del algoritmo propuesto.

4.1.1 Diseño de los algoritmos.

En esta propuesta se identifican a cada uno de los procesos con un identificador para realiza mejor la planificación y depende de si valor con un cero o no cero para

poder expandir los procesos hijos y realzar una planificación sencillo de implementar.

Algorithm 1 Planificador de procesos por abanico

```
Require: SocketUDP.h;
1: Crear.SocketUDP(host,port);
2: repeat
3:   for i=1 to NumProcesos  do
4:     Crear proceso Padre
5:     if ID ≠ 0 then
6:       sendMensaje();
7:       Crear.Proceso();
8:     else if ID = 0 then
9:       En espera
10:    end if
11:  end for
12: until Despachar todos los procesos
```

Figura 15. Algoritmo propuesto

La implementación del algoritmo en la figura 15 fue ejecutada en el laboratorio de supercómputo, utilizando instancias del sistema operativo reducido y un micro kernel llamado *Minix* versión 3. Cuenta con recursos básicos de un sistema operativo y es ideal para la ejecución de nuestra propuesta de algoritmos de planificación. Estas instancias fueron ejecutadas en un una herramienta para la administración de la nube de cómputo. La herramienta llamada OpenStack permitió la ejecución de un número considerable de instancias Minix, se realizaron pruebas de mínimo 1000 instancias y un manejo de 1000 procesos con su respectivo deadline.

Algorithm 63 Conexión de sockets usando procesos

```
1: if (child=fork())==-1 then
2:    perror("creando el proceso hijo");
      {este es el padre}
3:    close(sockets[0]);
4:    if read(sockets[1],buf,1024)<0 then
5:       perror("padre leyendo el mensaje ");
6:    end if
7:    print ("ID padre =%d mensaje de mi hijo -> %s ",getpid(),buf);
8:    if write(sockets[1],men2,strlen(men2)+1)<0 then
9:       perror("padre escribiendo el mensaje");
10:   end if
11:   print ("ID padre =%d respuesta -> %s ",getpid(),men2);
12:   close(sockets[1]);
13:   exit(1);
      {este es el hijo}
14:   close(sockets[1]);
15:   print ("ID hijo =%d pregunta -> %s",getpid(),men1);
16:   if write(sockets[0],men1,strlen(men1)+1)<0 then
17:      perror("hijo escribiendo mensaje");
18:   end if
19:   wait(&n);
20:   if read(sockets[0],buf,1024)<0 then
21:      perror("hijo leyedo mensaje");
22:   end if
23:   print ("ID hijo =%d respuesa de papa -> %s",getpid(),buf);
24:   close(sockets[0]);
25: end if
```

Figura 16. Procesos en Minix

En la figura 3 y línea de la figura 16. Se expresa el ciclo de ejecución de procesos que envían mensajes por medio del protocolo de comunicación remota Socket UDP. Ejecutado por cada uno de los peers instanciados en Minix alojados en la nube. Siendo estos *cloud-aware application* en OpenStack.

4.1.2 Desarrollo

En esta plataforma se tiene una sola entidad con tres elementos: nodo emisor, nodo receptor, nodo puente y nodo router. Se puede ver como un estado local, dependiendo de los requerimientos y restricciones para realizar la comunicación; los nodos pueden comunicarse con nodos vecinos por medio de una adecuada configuración de entidad, en un rango de transmisión de un emisor a un receptor en un vecindario L. La comunicación en este vecindario de configuración, describe una topología dinámica, ésta hace difícil el establecer y mantener una ruta de comunicación entre los nodos v_i v_j, teniendo en cuenta que $i \neq j$ y $v_i, v_j \in V$, tal que, en el grafo G=<P, R, A> sean aplicados a los nodos de una red móvil, presuntamente remotos por la poca disposición para la comunicación en un tiempo determinado futuro inmediato.

La plataforma permite detallar el funcionamiento de la comunicación entre dispositivos móviles. Nos permite crear una red de nodos como móviles y obtener un mejor funcionamiento entre la capa de sistema operativo y la plataforma necesaria para implementar un planificador de procesos bajo un ambiente virtual. En esta plataforma se pueden crear dispositivos móviles virtuales a partir de un evento, se debe conectar nodo por nodo y agregar el tiempo de creación como parte de un identificador.

El funcionamiento principal se basa en la comunicación de dispositivos móviles virtuales a través de un flujo de información entre dos capas, una gráfica y otra a bajo nivel. Gracias a esto se tiene la multiplataforma con JNI y el sistema operativo mínimo que utilizan los móviles es Minix [36]. Se utilizó Minix que es un sistema operativo por cacas con recursos mínimos gracias a la versatilidad.

```
#include <string.h>
#include <stdio.h>
#include <time.h>
#include <jni.h>
JNIEXPORT void JNICALL Java_DelayTime  (JNIEnv * env, jobject jobj,jstring i){
time_t rawtime;
struct tm * timeinfo;
time ( &rawtime );
timeinfo = localtime ( &rawtime );
const char *str = (*env)->GetStringUTFChars(env, i, 0);
printf("Dispositivo: 1  Grupo: A  Proceso: %s  Date: %s  \n",str,asctime (timeinfo));
}
```

Figura 17. Instrucción de la interfaz a bajo nivel

La interacción de dos lenguajes, como se ve en la figura 17, uno de alto nivel y otro de bajo nivel, para realizar una consulta de retardo en un algoritmo de planificación de procesos. Así como, el envío de la información del dispositivo móvil, como los tiempos de ejecución, de inicio, término y tiempo quantum, fueron asignados por el sistema así como, al proceso e identificador, etc. Esta información nos permite realizar la simulación en una supercomputadora en tiempo real y los resultados del planificador aplicado muestra un panorama completo [23].

5 Pruebas y resultados

Los resultados obtenidos están enfocados principalmente en la implementación de un algoritmo de planificación de tareas en una red P2P, tales respaldan a la toma de decisiones en grupos de agentes móviles en una comunidad de dispositivos móviles inalámbricos DMI, sobre una plataforma de vuelo con drones. Logrando la obtención de resultados en dispositivos móviles reales. Dando propuestas para mejorar el funcionamiento de robots móviles y dispositivos móviles. La aplicación de algoritmos y metodologías que se emplearon para realizar los resultados esperados fueron sencillos y con una buena respuesta de funcionamiento, basados en los resultados obtenidos se proponen nuevos algoritmos y metodologías con resultados para publicables en revistas científicas. Así también, se espera obtener una plataforma de medición de algoritmos de planificación capaz de medir los tiempos de respuesta, comprobando los algoritmos propuestos.

5.1 Pruebas

Una prueba de proyecto fue la aplicación de un registro atómico accedido por tres procesos p_1, p_2 y p_3 se representan en la figura 18 utilizando un diagrama clásico de espacio-tiempo. R.read () → v, significa que la operación de lectura correspondiente devuelve el valor v. En consecuencia, un observador externo ve la siguiente ejecución secuencial del registro R que cumple la definición de registro atómico:

R.write(1), R.read()→1, R.write(3), R.write(2), R.read()→2, R.read()→2.

Observemos que R.write(3) y R.write(2) son concurrentes, lo que significa que podrían aparecer ante un observador externo como si R.write(2) se hubiera ejecutado antes que R.write(3).

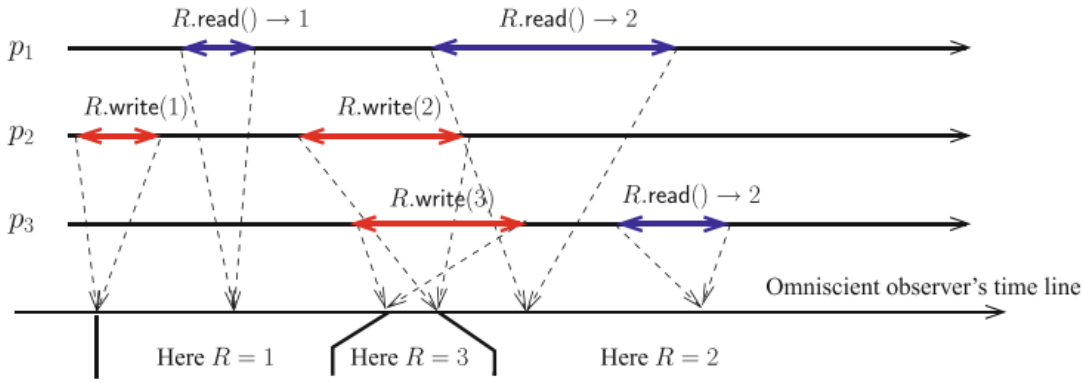

Figura 18. Una ejecución de registro atómico

Si este fuera el caso, la ejecución sería correcta si las dos últimas invocaciones de lectura (emitidas por p1 y p3) devuelven el valor 3; es decir, el observador externo debería ver la siguiente ejecución secuencial:

R.write(1), R.read()→1, R.write(2), R.write(3), R.read()→3, R.read()→3.

Observemos también que la segunda invocación de lectura por p1 es concurrente con R.write(2) y R.write(3). Esto significa que podría aparecer como ejecutado antes de estas dos operaciones de escritura o incluso entre ellas. Si parece haber sido ejecutado antes de estas dos operaciones de escritura, debe devolver el valor 1 para que el comportamiento del registro sea atómico.

Como se muestra en estos posibles escenarios (y como se ha notado antes), la concurrencia está íntimamente relacionada con el no-determinismo. No es posible predecir qué ejecución se producirá; solo es posible enumerar el conjunto de posibles ejecuciones que podrían producirse (solo podemos predecir que el que realmente se produce es uno de ellos).

Por qué la atomicidad es importante La atomicidad es un concepto fundamental porque permite la composición de objetos compartidos de forma gratuita (es decir, su composición no tiene costo adicional). Esto significa que, cuando se consideran

dos (o más) registros atómicos R1 y R2, el objeto compuesto [R1, R2] que se compone de R1 y R2 y proporciona los procesos con las cuatro operaciones R1.read (), R1.write (), R2.read () y R2.write () también son atómicos. Todo parece como si se ejecutara como máximo una operación a la vez, y la subsecuencia que incluye solo las operaciones en R1 es un comportamiento correcto de R1, y de manera similar para R2.

Esto es muy importante cuando uno tiene que razonar sobre un programa multiproceso cuyos procesos acceden a registros atómicos. Más precisamente, podemos seguir razonando secuencialmente sea cual sea el número de registros atómicos involucrados en un cálculo concurrente. La atomicidad nos permite razonar en un conjunto de registros atómicos como si fueran un solo objeto atómico "más grande". Por lo tanto, podemos razonar en términos de secuencias, no solo para cada registro atómico tomado por separado, sino también en todo el conjunto de registros como si fueran un solo objeto atómico.

5.2 Resultados

El algoritmos de planificaion de tareas como se muestra en la figura 19, Toma a cada proceso para identificar cúal esta listo para procesar un conjunto de tareas J_n, Entonces se crean los procesos $P_{m,}$ evitando entrara a un estado de error (E), asi como logarar su deadline.

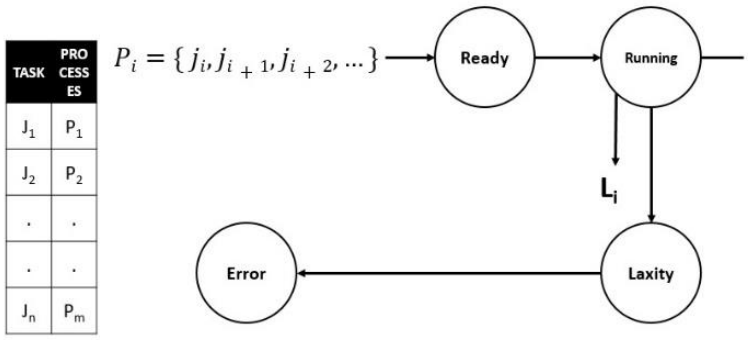

Figura 19. Algoritmo de planificación de tareas.

En la ecuacion 1, se expresa la condicion de: Si el tiempo de latencia (tiempo perdido) X_i de cada uno de los procesos es mayor que la suma de estos en un timpo amsoluto restringido (d_i), entonces el error tiende al fallo y lso proceso terminaran esperando.

$$E = \sum_{i=1}^{n}(d_i) \leq X_i \quad (1)$$

donde:

X_i: Tiempo en que la tarea puede ser retrasada en sus activaciones para completar su ddeadline.

L_i: retardo de j_i

d_i: deadline absoluto.

Teniendo un planificador de tareas en tiempo real σ(t) cuyos procesos se generan en un árbol de expansión, como un conjunto de procesos P, un conjunto de recursos R y un conjunto de tareas J, como se muestra en la ecuación (2), se espera Para evitar el problema de planificación que es el efecto dominó. Suponiendo que la solución de utilizar un gráfico acíclico relacionado reduzca estos errores en la planificación, como se propuso en [23].

$$G < P, R, J, A > \; p_i \in P \,,\; r_k \in R \;\; y\, j_l \in J \quad (2)$$

donde:

G: es un grafo aciclico conexo

A: Aristas de vértices procesos a vértices recuros y vértices tareas

Las ejecuciones se realizaron en un nodo asignado por el laboratorio de supercomputadoras LNS-BUAP, utilizando instancias del sistema operativo reducido con un micro-kernel llamado Minix versión 3. El micro-kernel tiene recursos básicos de un sistema operativo, siendo ideal para la ejecución De los algoritmos de planificación. Estos casos se llevaron a cabo con una herramienta para la administración de la nube de cómputo. La herramienta llamada OpenStack permitió la ejecución de un número considerable de instancias de Minix, se realizaron varias pruebas con al menos 1,000 instancias y un manejo de 1,000 procesos.

Según el funcionamiento del algoritmo del abanico; cuya implementación lanza mil hilos que ejecutan una tarea para cada proceso creado en un rango de ochocientos procesos y en conjunto con el algoritmo de Lamport como se usa en [23]; primero se obtiene el número del giro del hilo, que se calcula como el máximo de los giros de los otros hilos, más uno. Antes de que el hilo pueda entrar en su sección crítica, se asegura que se obtenga el número más bajo.

Como resultado de la aplicación del algoritmo de Lamport, se obtuvo una reescritura del código original del algoritmo del abanico, la fecha límite exitosa de una gran parte de los procesos creados y sus hilos demostrados están garantizados con sus tiempos de ejecución. En la gráfica 20, puede ver cómo, por secciones de cien procesos, los tiempos de ejecución de las tareas aumentan, siendo la sección de P701 a P800 la que tiene los tiempos de ejecución más altos. Debido a la planificación realizada por el algoritmo de Lamport.

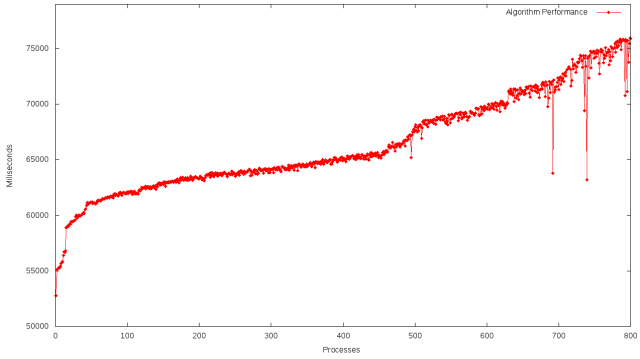

Fig. 20. Processes with deadline.

Durante las pruebas realizadas, una vez que se iniciaron todos los procesos, se midió el tiempo para alcanzar su fecha límite. Como ejemplo de estas pruebas, en la Fig. 2 se muestra que el 92% de los procesos alcanzaron su fecha límite a priori en menos de un minuto; Mientras que los procesos restantes alcanzaron su fecha límite en el lapso de dos horas.

Se deben ejecutar más pruebas para confirmar que se trata de un comportamiento promedio y también para analizar qué sucede cuando aumenta la cantidad de procesos y recursos. Sin embargo, el algoritmo propuesto generó resultados que pueden usarse para compararlos con los resultados de otros algoritmos similares para determinar si reduce o no el tiempo para la programación de procesos.

La Figura 20 se muestra los procesos medidos en segundos y es posible apreciar los resultados obtenidos por las pruebas realizadas en el nodo de laboratorio LNS. Cabe destacar que el 92% de los procesos alcanzaron su fecha límite establecida a priori. A partir de 980 procesos, se observa una constante con respecto al tiempo. La interpretación de esta declaración indica que el programador tiene una breve mejora con respecto a la fecha límite de los procesos. Después de un tiempo de procesamiento, se logra el 98.9% de los procesos que llegan a su fecha límite. El 8% restante logró una mejora considerable en el rango de 1.8s a 2 segundos. Y se mantiene con este comportamiento. Finalmente, se esperan mejoras para este algoritmo propuesto que aumenta la cantidad de procesos y recursos.

En un sistema distribuido móvil (MDS), hay nodos como un sistema dinámico, por lo tanto, es necesario realizar el análisis de la métrica que se utilizará, que es la minimización de la suma del peso ponderado en tiempos determinados, basado en un costo de enviar un paquete entre dos nodos, como se expresa en la ecuación (2).

Esta métrica es importante ya que en el MDS la diferencia de valores se aplica a los sistemas en tiempo real, propuestos como en [8 y 5]. También se propone una métrica, para comparar diferentes algoritmos basados en teoremas de programación de procesos en uniprocesadores y multiprocesadores, midiendo el tiempo de cálculo requerido para la determinación de un planificador que satisfaga el orden parcial y las restricciones de recursos.

$$\mathrm{M_c} = \frac{1}{\mathrm{n}} \sum_{i=1}^{n-1} \mathrm{N_{P_i}}(\mathrm{t_c}) - \mathrm{N_{p_{(i+1)}}}(\mathrm{t}_c) \qquad (2)$$

$$t_c = \max(f_i) - \min(a_i)$$

where:

M_c	proposed metric.
$N_{pi}(t_c)$	node weighting i
t_c	total completion time
f_i	end time of tasks
a_i	start time of the task

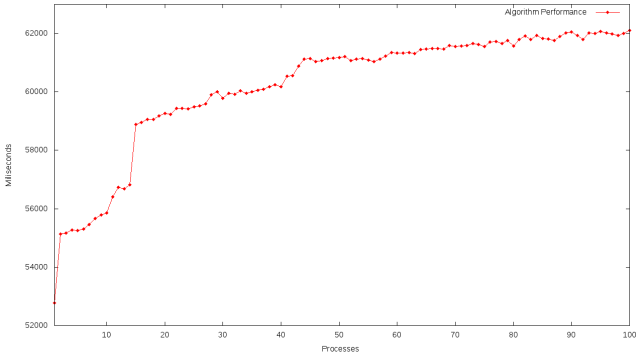

Fig. 21. Primera etapa de la ejecucion de los procesos

La figura 21 muestra el inicio de las ejecuciones de las tareas de 1 a 100 procesos. Los primeros procesos entran en el plazo en buenas condiciones. A medida que aumenta el número de procesos y, a su vez, el tiempo para alcanzar su fecha límite.

Conclusiones.

En esta propuesta de investigación, se estudiaron los algoritmos de planificación de tareas en consenso y en tiempo real, así como las técnicas, métodos y algoritmos de los sistemas móviles distribuidos, estableciendo el estado del arte centrado en las aplicaciones e investigaciones recientes. Además de proponer una metodología y una métrica capaces de satisfacer los objetivos y la meta presentada. Los modelos autónomos se propusieron enfocar e integrar las operaciones del planificador de procesos con operaciones enfocadas en la solución de problemas, lo que nos permite obtener un control adecuado en el cálculo de los tiempos de demora en un sistema. Finalmente, se realizó la simulación de algoritmos y métodos en un laboratorio de computación de alto rendimiento.

En este proyecto se estudiaron los algoritmos de planeación de procesos basados en consenso, así como, las técnicas, métodos y algoritmos de los sistemas en tiempo real y sistemas distribuidos móviles, estableciendo el estado del arte enfocado a las aplicaciones e investigaciones reciente, además de proponer una metodología y una métrica capaz de satisfacer los objetivos y la meta presentados. Los modelos autónomos se propusieron para enfocar e integrar operaciones del planificador de procesos como el EDF, con operaciones enfocadas a la solución de problemas que nos permitan obtener un control adecuado en calcular los tiempos de retardo en un sistema. Por último se planteó la simulación de los algoritmos y métodos en un laboratorio de cómputo de alto rendimiento y la aplicación en sistemas en tiempo real, como lo son dispositivos móviles, drones o celulares.

Bibliografía.

1. Luis Angel Zamarripa-Almazan. Tesis:. Desarrollo y diseño de una plataforma visual para la aplicación de algoritmos de planificación de procesos en tiempo real “JScheduling tool”. FCC-BUAP. Octubre 2017.
2. Mariano Larios-Gómez, Luis Angel Zamarripa-Almazan. JScheduling: A graphical interface for applying a process scheduling algorithm. Special issue of the Research in Computing Science Journal, ISSN 1870-4069 (Indexed by DBLP, LatIndex, Periodica). 2017.
3. M. Larios-Gómez, G. Trinidad-García, H. Salazar-Martínez, E. A. Hernández-Vicenttin, A. Almada-Díaz. A basic algorithm for real-time processes scheduling. XIV Congreso Internacional sobre Innovación y Desarrollo Tecnológico CIINDET 2018. Cuernavaca Morelos, México.
4. M. Scott Corson; J. Macker; S.G. Batsell. Architectural considerations for mobile mesh networking. Military Communications Conference. MILCOM '96, Conference Proceedings, IEEE.
5. J.A. Stankovic ; M. Spuri ; M. Di Natale ; G.C. Buttazzo. Implications of classical scheduling results for real-time systems. Browse Journals & Magazines Computer. Volume: 28 Issue: 6, 1995
6. Sengul Cigdem and Robin Kravets. Bypass routing: An on-deman local recovery protocol for Ad Hoc netwoks. Volume 4 Issue 3, May, 2006 Pages 380-397 Elsevier Science Publishers B. V. Amsterdam, The Netherlands, The Netherlands.
7. V.D. Park; M.S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution. Proceedings IEEE.
8. C. Lu ; J.A. Stankovic ; G. Tao ; S.H. Son. Design and evaluation of a feedback control EDF scheduling algorithm. Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No.99CB37054).
9. Gómez, J. A. H., & Pérez, H. B. (2015). Global Scheduling of Confined Tasks Based on Consensus. IEEE Latin America Transactions, 13(3), 825-834.
10. Stankovic, J. A. (1985). An Application of Bayesian Decision Theory to. IEEE Transactions on Computers, 100(34).
11. Pearl, J. (2014). Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann.
12. Ramasubramanian, V., Haas, Z. J., & Sirer, E. G. (2003). SHARP: A hybrid adaptive routing protocol for mobile ad hoc networks. In Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing (pp. 303-314). ACM.
13. Hui, Q., & Haddad, W. M. (2008). Distributed nonlinear control algorithms for network consensus. 44(9), pp. 2375-2381. Automatica.

14. Hong, Y., Chen, G., & Bushnell, L. (2008). Distributed observers design for leader-following control of multi-agent networks. Automatica, 44(3), pp. 846-850.
15. Andrew S. Tanenbaum: The Impact of MINIX. Computer. Page(s): 7 - 8 Volume: 47, Issue: 7, July 2014.
16. DCyTIC. (2016). Especificaciones de la supercomputadora. 2017, de Laboratorio Nacional de Supercómputo del Sureste de México Sitio web: http://www.lns.org.mx/infrastructura
17. Kshemkalyani, A. D., & Singhal, M. (2011). Distributed computing: principles, algorithms, and systems. Capítulo 14. Cambridge University Press.
18. Esquivel-Flores, O., Benítez-Pérez, H., & Ortega-Arjona, J. (2012). System Based Upon Scheduling Strategy Using Numerical Simulations. Issues on Communication Network Control. INTECH Open Access Publisher 2016.
19. Benitez Hector, García-Nocetti, Fabián. Reconfigurable Distributed Control. 10.1007/1-84628-196-2
20. Oriol Castillo, H. Benítez-Pérez. A Novel Technique to Enlarge the Maximum Allowable Delay Bound in Sampled-Data Systems , Congreso Nacional de Control Automático 2017 Monterrey, Nuevo León, Mexico, Octubre 4-6, 2017.
21. Paul Mendez-Monroy and Hector Benitez-Perez. Supervisory fuzzy control for networked. ICIC International °c 2009 ISSN 1881-803X. Control systems. Volume 3, Number 2, June 2009
22. Mancina, A., Faggioli, D., Lipari, G. et al. Real-Time System (2009) 43: 177. https://doi.org/10.1007/s11241-009-9086-5.
23. M. Larios et all. Scheduling: A graphical interface for applying a process scheduling algorithm. Research in Computing Science, Vol. 145 Page(s) 119-126. ISSN 1870-4069. Applications of Language & Knowledge Engineering 2017.
24. Cortés, J. (2008). Distributed algorithms for reaching consensus on general functions. 44(3), pp. 726-737. Automatica.
25. M. Larios. Programación concurrente y paralela: un enfoque práctico. Editorial BUAP. 2012
26. Hui, Q., & Haddad, W. M. (2008). Distributed nonlinear control algorithms for network consensus. 44(9), pp. 2375-2381. Automatica.
27. Hong, Y., Chen, G., & Bushnell, L. (2008). Distributed observers design for leader-following control of multi-agent networks. Automatica, 44(3), pp. 846-850
28. Introduction to Proxy Servers in Computer Networking. Obtenido de Lifewire [Consulta:17 de octubre de 2016], Sitio web: https://www.lifewire.com/introduction-to-proxy-servers-computer-networking-816448

29. Burande A., P. A. Wireless Network Security by SSH Tunneling. International Journal of Scientific and Research Publications, 4-4. (2014).
30. Globus.org. Documentation. [Consulta:17 de octubre de 2016]. Sitio web: http://toolkit.globus.org/toolkit/docs/4.1/admin/docbook/gtadmin-env-var.html
31. Doug Lea. Concurrent Programming in Java. Design Principes and Patterns. Sun Microsystem, Addison Wesley (2003).
32. Bill Lewis, Daniel J. Berg. Multithreaded Multiprogramming in JAVA. Sun Microsystem Pearson Prentice-Hall(1996).
33. M. L. Liu. Distributed Computing: Principles and Applications, Addison Wesley 2004.
34. Vijay K. Garg. Concurrent and Distributed Computing in Java. Wiley Inter-Science (2004).
35. Cay S. Horstman, Gary Cornell. Core Java 2. Volumen 2. Sun Microsystem Pearson Pretinces-Hall (2005).
36. Andrew S. Tanenbaum. Sistemas operativos, Minix. Person (2015).